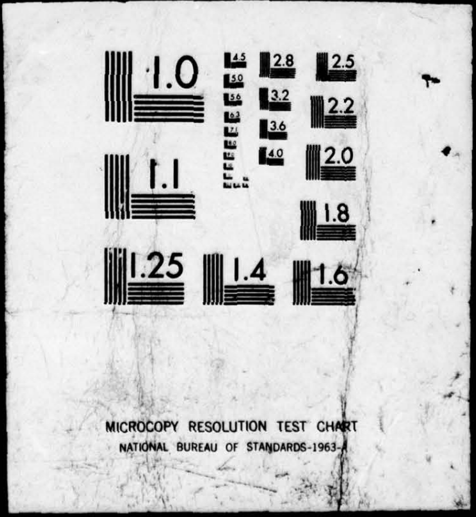
AD-A035 337 STANFORD RESEARCH INST MENLO PARK CALIF AUGMENTATION --ETC F/G 17/2
TRANSMISSION CONTROL PROTOCOL SPECIFICATION.(U)
JUL 76 J B POSTEL, L L GARLICK, R ROM DCA100-76-C-0034
UNCLASSIFIED SRI-ARC-35938 NL

1 OF 2
AD
A035337

SIFIED

1 OF 2

AD
A035337



STANFORD RESEARCH INSTITUTE
Menlo Park California 94025 U S A

2

Transmission Control Protocol Specification

See 1473 in back

15-July-76

Jonathan B. Postel
Larry L. Garlick
Raphael Rom

Augmentation Research Center

Stanford Research Institute
Menlo Park, California 94025

(415) 326-6200

This report is the deliverable "Transmission Control Protocol Specification (Final)" as specified in contract DCA100-76-C-0034.

new

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE — READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) (6) Transmission Control Protocol Specification. (9) Final rept.,		(14) SRI-ARC-35938, SRI-ARC-35939 5. TYPE OF REPORT & PERIOD COVERED 6. PERFORMING ORG. REPORT NUMBER 35938/35939
7. AUTHOR(s) (10) [redacted] Jonathan B. Postel, [redacted] L. Larry Garlick, [redacted] Raphael [redacted] Rom		8. CONTRACT OR GRANT NUMBER(s) (15) DCA 100-76-C-0034 New
9. PERFORMING ORGANIZATION NAME AND ADDRESS Augmentation Research Center Stanford Research Institute Menlo Park, California 94025		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS DCA/Defense Communications Engineering Center 1860 Wiehle Avenue, Reston, Virginia 22090 ATTN: Code R850		12. REPORT DATE (11) 15 Jul[redacted]76 13. NUMBER OF PAGES 181 (12) 183p.
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) N/A		15. SECURITY CLASS. (of this report) Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for Public Release; Distribution Unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

N/A

18. SUPPLEMENTARY NOTES

N/A

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Descriptors: Communication Networks, Data Transmission Systems.

Identifiers: Computer Networks.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This is the specification of the Transmission Control Protocol (TCP) which is the host-to-host protocol for the AUTODIN II network.

The AUTODIN II system provides the capability for geographically distri buted computers, called hosts, to communicate with each other. The hosts a a diverse set of computers of differing manufacture, speed, word size, and operating system. The AUTODIN II system provides a mechanism for communic between hosts; the TCP specifies how the hosts use this mechanism to prov communication services to processes and ultimately to human users. CONT

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (W

The reader of this document is assumed to be familiar with the concepts of the AUTODIN II system, and with operating system concepts. In particular, the reader is expected to have read the AUTODIN II Specification.

It is intended that the information presented here be sufficient and complete enough to allow a competent systems programmer to implement an operating system program module to carry out this protocol.

In the following sections a brief background of the AUTODIN II network, and the protocol environment within it, is presented. This introduces the more detailed specification in the sections on TCP function, environment, and implementation. The document includes appendices on several details of TCP implementation. (modified author abstract)

A

PREFACE

This report is the deliverable "Type "A" Specification of the Transmission Control Protocol (Final) for AUTODIN II as specified in contract DCA100-76-C-0034.

ACKNOWLEDGEMENTS

We wish to extend our thanks to Vint Cerf and his colleagues at the Digital Systems Laboratory of Stanford University for their help in acquainting us with some of the technical issues of TCP implementation, and readily discussing new design and implementation concepts and suggestions that we had to offer.

We also would like to recognize the efforts of Ray Tomlinson of Bolt, Beranek, and Newman in providing helpful insights into the resynchronization problem.

James White and David Maynard of SRI-ARC were very helpful in evaluating this document in its final stages.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF APPENDICES

1. INTRODUCTION

1.1. This is the specification of the Transmission Control Protocol (TCP) which is the host to host protocol for the AUTODIN II network.

1.2. The AUTODIN II system provides the capability for geographically distributed computers, called hosts, to communicate with each other. The hosts are a diverse set of computers of differing manufacture, speed, word size, and operating system. The AUTODIN II system provides a mechanism for communication between hosts; the TCP specifies how the hosts use this mechanism to provide communication services to processes and ultimately to human users.

1.3. The reader of this document is assumed to be familiar with the concepts of the AUTODIN II system, and with operating system concepts. In particular the reader is expected to have read the AUTODIN II Specification [reference 1].

1.4. It is intended that the information presented here be sufficient and complete enough to allow a competent systems programmer to implement an operating system program module to carry out this protocol.

1.5. In the following sections a brief background of the AUTODIN II network, and the protocol environment within it, is presented. This introduces the more detailed specification in the sections on TCP function, environment, and implementation. The document includes appendices on several details of TCP implementation.

2. GLOSSARY

2.1. acceptability filter--a dynamic range in the TCP sequence number space used to detect duplicates and segments with illegal sequence numbers.

2.2. access circuit--an addressable hardware port on the LCM; the point at which communication lines for local terminal devices, TAC's, and hosts are attached.

2.3. ARC--Augmentation Research Center of SRI.

2.4. ASCII--American Standard Code for Information Interchange; a seven bit encoding of textual characters.

2.5. authorization table--a table used by the TCP to check S/P/T values during connection management.

2.6. BSL--Binary Segment Leader; a S-segment header used for PS-SIP communication.

2.7. CCU--Channel Control Unit; an MCCU or SCCU.

2.8. EOL--End-of-letter flag; used to indicate that a segment includes the end of a letter.

2.9. FED--Front-End Device; a method of connecting a host to a PS with minimum alteration to the host operating system to support protocol functions.

2.10. host--a computer connected to a network that executes programs on behalf of its users but does not necessarily offer services to the other computers on the network.

2.11. host-entity--a host, FED, CCU, or TAC.

2.12. HSI--Host Specific Interface; a sub-module of the CCU that interfaces the TCP to the host.

2.13. ISN--Initial Sequence Number; the first sequence number used when a connection is first synchronized or resynchronized.

2.14. LCM--Line Control Module; controls communication between a PS and hosts, terminal equipment, TAC's, and remote PS's.

2.15. LCN--Local Connection Name; a handle that is given to the user as a shorthand identifier for a TCP connection.

2.16. LISTEN--a type of connection opening in which some part of the destination address, send precedence, or TCC has been unspecified. This type of open allows a user or process to wait indefinitely until a connection attempt is made by a remote user, and is thus useful for open synchronization and server processes.

2.17. MCCU--Multiple Channel Control Unit; allows the interfacing of a host to a PS and provides up to 32 connections with remote hosts or subscribers.

2.18. NCC--Network Control Center.

2.19. NVT--Network Virtual Terminal; a standard, network-wide, intermediate representation of a canonical terminal.

2.20. octet--eight bits.

2.21. Periods Processing--a temporary mode of operation assumed when host resources are required for special processing of highly restricted information.

2.22. port--a name chosen from a universal name space (not necessarily unique, however) that identifies the stream of information passed between a process and the network.

2.23. PS--Packet Switch of the AUTODIN II network.

2.24. PSN--packet switch network.

2.25. S-segment--the concatenation of a Binary Segment Leader and a T-segment.

2.26. S/P/T--Security, precedence, and TCC.

2.27. SCCU--Single Channel Control Unit; allows the interfacing of a host to a PS with a single connection capability.

2.28. SIP--Segment Interface Protocol (or the program that implements it); a sub-module of the host, TAC, or CCU that interfaces to the LCM for PS-TCP communication.

2.29. socket--an entity defining one end of a TCP connection; the inter-network-wide name of a process port which is a concatenation of network identifier, TCP identifier, and port identifier.

2.30. SRI--Stanford Research Institute, Menlo Park, California.

2.31. subscriber--the logical address of an access circuit, which at the PS level represents an end-user or a host computer.

2.32. switch directory--a table containing address translation tables and S/P/T control information that is used by the switch to direct and restrict traffic to/from an access circuit.

2.33. T-segment--the concatenation of a TCP-TCP header and data (optional).

2.34. TAC--Terminal Access Control module. Consists of the following sub-modules: TC, THP, TCP, and SIP.

2.35. TC--Terminal Control module; a sub-module of the TAC that interfaces directly to the LCM for user-TAC communication.

2.36. TCB--Transmission Control Block; a bookkeeping block used by the TCP(program) to maintain information about a full duplex TCP connection.

2.37. TCC--Transmission Control Code; a user group code that serves to compartmentalize traffic and define controlled communities of interest among subscribers.

2.38. TCP--Transmission Control Protocol (or the program that implements it).

2.39. Telnet--an Arpanet protocol (or the program that implements it) that specifies the communication interaction such that a user on a terminal of one computer gains access to the services of another computer as if she were a local user of the second computer.

2.40. THP--Terminal-to-Host Protocol (or the program that implements it).

2.41. window--a dynamic range in the sequence number space used for flow control.

3. BACKGROUND

3.1. Configuration Overview

3.1.1. The AUTODIN II system is a packet switched communication network linking a wide range of terminals and host computers. There are four main components of a packet switched communication system:

3.1.1.1. The Backbone: the packet switches and their interconnecting trunk lines.

3.1.1.2. The User Interface: the various interface devices that connect to subscriber equipment. Examples of the interface devices are the Terminal Access Controller (TAC), the Multiple Channel Control Unit (MCCU), and the Single Channel Control Unit (SCCU). Figure 1 shows the various user interfaces.

3.1.1.3. Protocols and conventions needed for communications and control between the various components in the backbone.

3.1.1.4. Protocols and conventions needed for communications and control between the various processes in the user environment. These include the host level protocol (TCP) and the terminal access protocol (THP).

3.1.2. AUTODIN II is designed to provide a communication service needed to support Interactive, Query/Response, Bulk Data Transfer, and Narrative applications to meet the man-to-man, man-to-computer, and computer-to-computer data transmission requirements of DoD users in CONUS and certain overseas subscribers.

3.1.3. Host computers are defined as computers, or front-end devices (FED) associated with computers, that are capable of simultaneously conducting multiple conversations with other hosts or terminals.

3.1.4. In addition, two types of host interface control units are provided: a SCCU and a MCCU. The purpose of these two interface control units is to allow certain host subscribers to interface the network with little or no modification to their host hardware or software.

3.1.5. There are two program modules which reside either in user hosts, or in user-provided network front-end devices (FED), or in

SUBSCRIBER INTERFACE CONFIGURATIONS

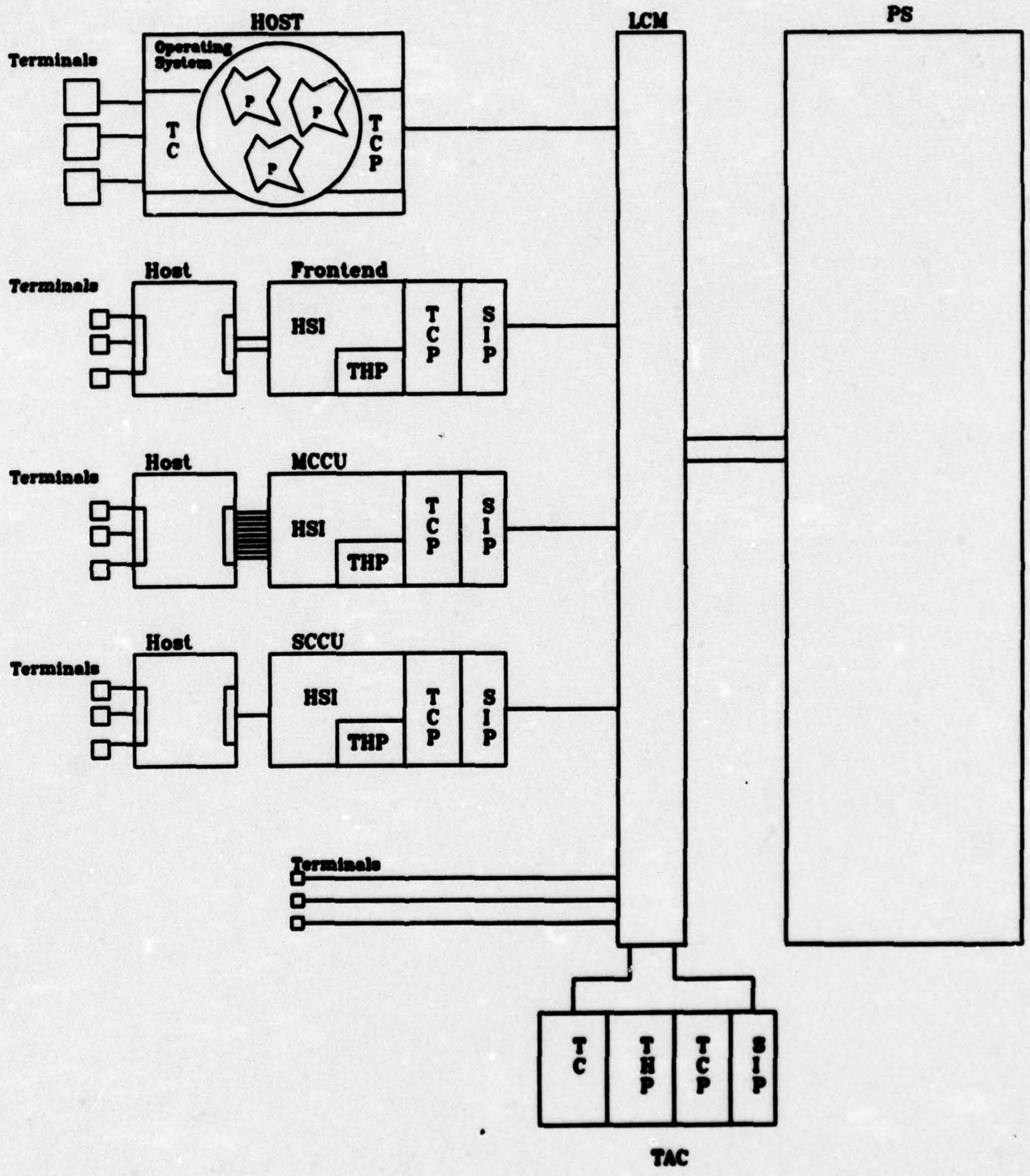


Figure 1

the SCCU's and MCCU's provided by the AUTODIN II. These program modules are used to access the network: one program module is the Transmission Control Program (TCP), which performs host-to-host protocol functions; the second program module is the Segment Interface Protocol (SIP), which performs the network interface protocol functions.

3.1.6. Terminals are defined as character oriented devices capable of conducting communications with only one destination at a time. Terminals shall access the network through a TCP-SIP configuration using a Terminal-to-Host Protocol (THP). This protocol, which is similar to the ARPANET Telnet, interfaces the terminals to the TCP program module in the TAC to provide the mechanism for terminals to access foreign host computers via the communications network.

3.1.7. For terminals associated with a local host computer, the THP/TCP/SIP program modules reside in the host, FED, SCCU, or MCCU as the case may be. For terminals not associated with a local host computer the THP/TCP/SIP program modules reside in the Terminal Access Control (TAC) functional module associated with the Packet Switch (PS).

3.1.8. Subscriber Interface Configurations

3.1.8.1. The above mentioned modules must be tailored to fit into a variety of user configurations. The configurations are shown in Figure 1 and described below.

3.1.8.1.1. Host

The most general configuration is the direct host configuration, in which the TCP and the THP are implemented in the host as operating system or service process functions.

3.1.8.1.2. Host Front-End

The next configuration shown, the Front-End Device (FED) case, requires a host specific interface (HSI) software module in the front-end which implements still another protocol between the front-end and the Host.

In most cases the front-end is intended to deal with the complications of the network protocol, saving the Host both CPU cycles and memory space. The protocol used between the Host and the front-end is intended to be as simple as possible, often a simulation of some device the Host is otherwise interfaced with.

3.1.8.1.3. MCCU

The third case is the MCCU which is intended to interface a host to the network with a minimal effect on the Host. The strategy is much the same as the front-end except that the physical interface is defined to be a set of up to 32 lines where each line is associated with one logical connection at a time. The intent is that each line may appear to the host as a terminal or some other simple device.

3.1.8.1.4. SCCU

The next case, the SCCU, provides a single-connection interface to the network. The principal intended use of the SCCU is to provide to a pair of hosts currently connected via a private line the less expensive alternative of using the network. In this situation each host would have an SCCU interface to the network. Other uses of the SCCU might include an interface for a synchronous remote job entry station. Implementation details for a SCCU are significantly simpler than for the other configurations. A separate appendix, Appendix I, contains a list of all the simplifications realizable with the SCCU configuration.

3.1.8.1.5. TAC and Terminals

The fifth case is the TAC. The TAC is intended to provide the TC, THP, TCP, and SIP services for the terminals directly connected to the LCM.

3.2. Protocol Overview

3.2.1. Before discussing the particulars of the TCP, a description of the protocol environment is presented. First a scenario of a message transmission is outlined, then the protocol functions are discussed, and finally the structural relationship between the protocols is described.

3.2.2. Message Transmission Scenario

3.2.2.1. The major modules in the communication path between a user and a service process are shown in Figure 2.

PROTOCOL ENVIRONMENT

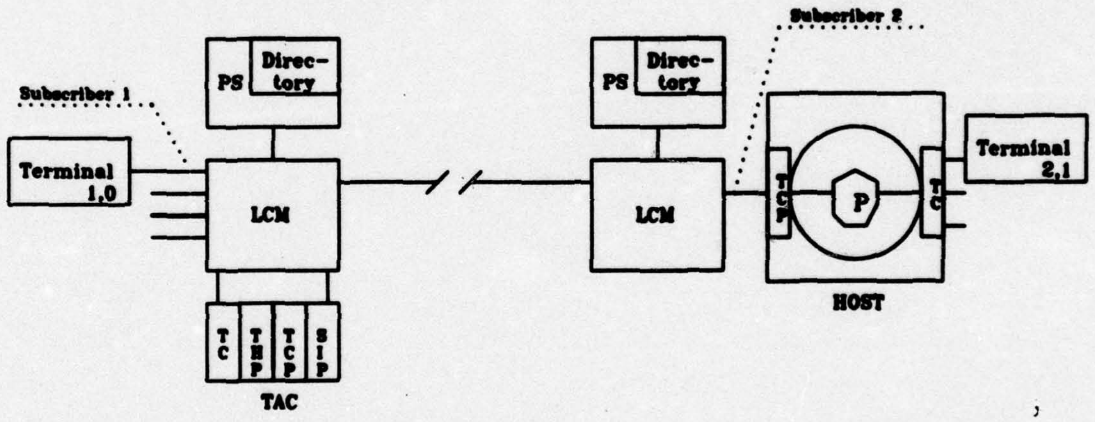


Figure 2

PROTOCOL ENVIRONMENT

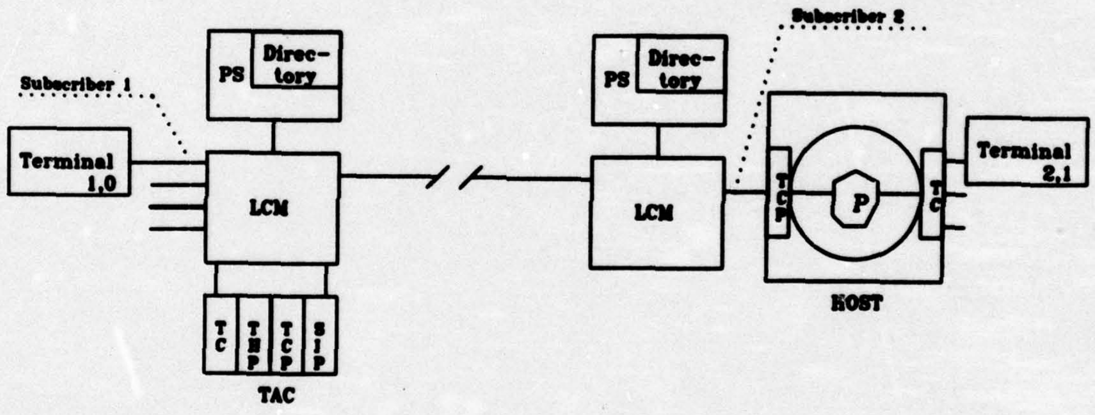


Figure 2

3.2.2.2. The user (Subscriber 1) at Terminal 1,0 first enters information about the destination she wishes to address, and the S/P/T parameters for the communication. The user's input is routed through the Line Control Module (LCM) to the TC module in the TAC. The TC delivers the data to the THP, which processes the information. The THP makes an OPEN request on the TCP specifying the destination and S/P/T as indicated by the user. The TCP sets up a table entry (a TCB) for the pending connection, but no network messages are transmitted.

3.2.2.3. When the user enters her first data message, the THP packages the text into a letter and passes the letter to the TCP with a SEND request. The TCP checks the TCB entry for this connection and finds that the connection is not yet established. The TCP then exchanges connection establishing messages with the distant TCP. The TCP passes the messages (segments) to the SIP, which forwards the segment to the PS via the LCM.

3.2.2.4. When this opening exchange is completed, the TCP reformats the original letter into segments which include sequencing and the S/P/T control information. As with all segments destined for a foreign TCP, this segment is passed to the SIP for forwarding through the network.

3.2.2.5. The PS verifies that the S/P/T values indicated in this message are valid by checking the Directory entry for this subscriber. On passing this test the PS routes the segment to another PS, and by travelling from one PS to another the segment eventually reaches the PS of the destination address.

3.2.2.6. At the destination the segment is routed from the PS through the LCM to the Host (in our example). In the Host the segment is processed by the TCP. The TCP reformats the segment, deleting the control information and possibly combining several segments to form a letter for delivery to the destination process. The destination TCP sends an acknowledgement (possibly combined with data) to the data originating TCP. The destination process acquires the data as the result of a RECEIVE request.

3.2.3. Protocol Functions

3.2.3.1. In this scenario several functions have been mentioned or implied. In the following we will review the functions and indicate the division of responsibility for implementing them among the protocol modules.

3.2.3.1.1. Conceptual Model for Higher Level User

Each protocol module provides a model of its communication facility to the next higher level of use. This model is an attempt to make the lower levels of protocol invisible to the user of the particular protocol module. Examples of these models are the inter-process communication facility provided by the TCP and the terminal-to-process model provided by the THP.

3.2.3.1.2. Device Dependent Terminal Control

The device dependent terminal characteristics include conversion of character coding, folding of long lines of text, etc. This function is performed by the TC.

3.2.3.1.3. Character Set Standardization

The community of users of the network utilize many different types of terminals and host computer systems. Among these terminals and systems several character sets are used. To enable users to interact with all other subscribers, a network standard character set is defined. The THP is responsible for any translation necessary.

3.2.3.1.4. Interaction Mode Control

The control by the user over the mode of interaction is accomplished via an interaction with the THP. These modes include the choice between character-at-a-time and line-at-a-time interaction, and between local echoing and remote echoing.

3.2.3.1.5. User Command Interpretation

The user should have the ability to control the interaction mode, connection opening and closing, and various options possible under THP as well as the device characteristics. These controls should be indicated through the use of a command language. The THP is responsible for parsing the command language and carrying out the requested action, often by calling on the TC or TCP.

3.2.3.1.6. Multiplexing Connections

The use of the network by a host or TAC must be subdivided or multiplexed into many conversations so that the many users (terminals or processes) are able to simultaneously obtain services. The TCP provides this multiplexing function by presenting to the THP level the conceptual notion of connections, which is the basic mechanism for process-process communication. Connections may be thought of as logical circuits.

3.2.3.1.7. Ordering

Most applications of the network require data to be delivered in the order in which it is sent. This function is accomplished in the TCP by means of sequence numbers.

3.2.3.1.8. Packaging

The users usually wish to be spared the details of segment formats and network communication. A packaging function is performed by each module along the path to provide the user with a simpler view of the communication system. The THP provides for packaging of user text into letters and vice versa. The TCP provides for the packaging of letters into segments and vice versa.

3.2.3.1.9. Reliable Transmission

Users expect that all messages they present to the network will be delivered. To provide for the reliable transmission of messages both the PS and the TCP utilize sequence numbers, positive acknowledgements, retransmission, and flow control mechanisms.

3.2.3.1.10. Security, Precedence, and User Group Monitoring

The security, precedence, and user group of each message must be verified. The PS will do this by checking each message against the PS Directory at both the source and destination. The TCP may provide additional checking.

3.2.3.1.11. Multiplexing Hosts and Subscribers

The backbone network is shared among many subscribers by multiplexing the message flow according to the subscriber identification. Subscribers may be hosts or terminals connected to TAC's; they are not processes. This function is provided by the PS.

3.2.3.1.12. Addressing and Routing

The PS provides addressing for Hosts and most terminals by means of the concept of Subscribers. The TCP provides addressing for processes and other terminal users with sockets.

The PS attempts to provide for the optimal routing of messages through the network.

3.2.4. Levels of Protocol

3.2.4.1. The protocol modules form a hierarchy, as shown in Figure 3. Users conceptually transmit text to other users. In actuality they exchange text with the supporting THP's. The THP's in turn conceptually exchange letters, but actually send and receive letters to and from the supporting TCP's. The TCP's conceptually exchange T-segments, but in fact these T-segments are transmitted via the SIP's to the PS's as S-segments. The PS's actually exchange packets.

NLS graphics

LEVELS OF PROTOCOL

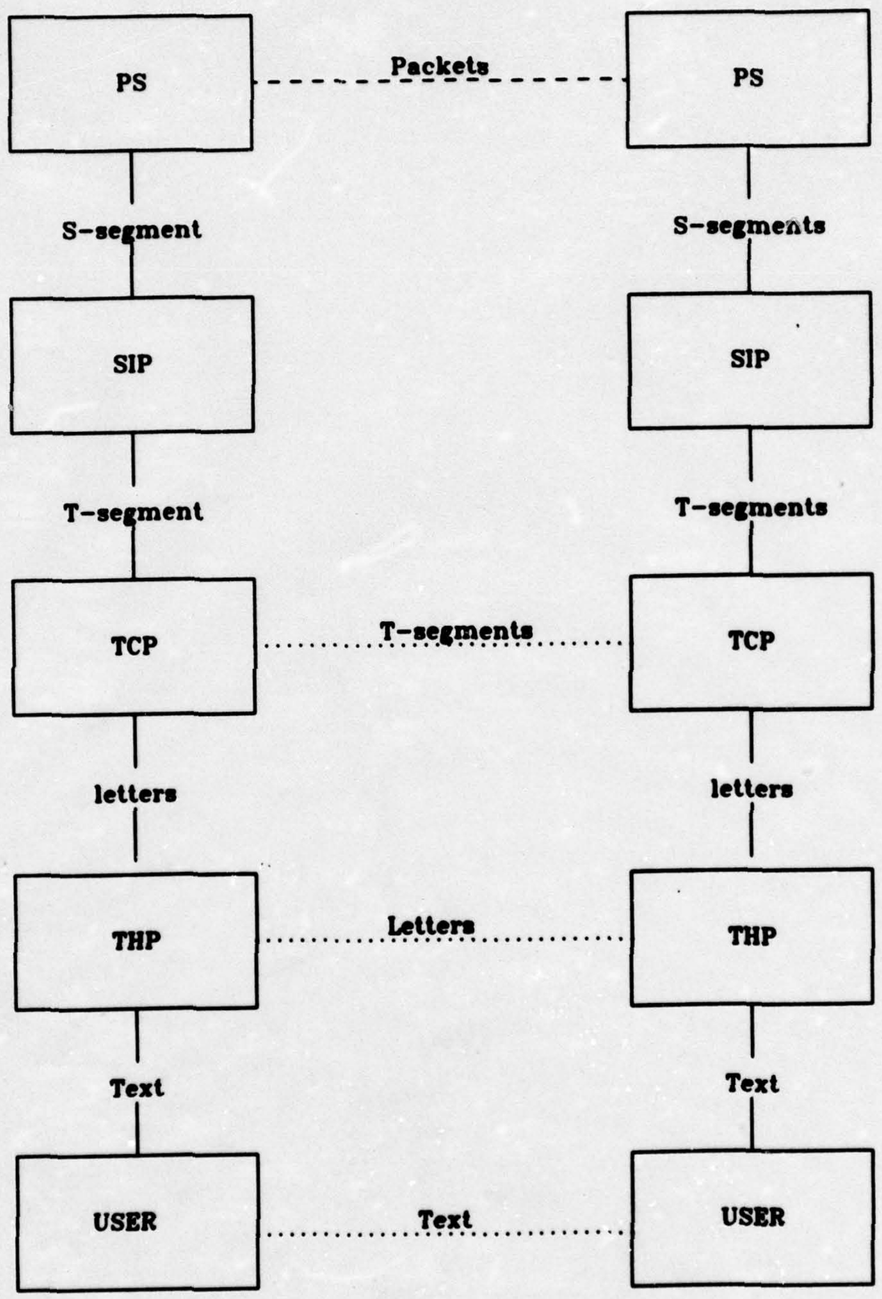


Figure 3

4. TCP FUNCTION

4.1. In the description of protocol functions the TCP has been assigned the tasks of handling multiple connections of varying precedences and security levels, packaging and ordering users' letters, and generally providing the user with reliable transmission. These functions are elaborated upon below, where they are grouped into three new categories--connection management, information transfer management, and security, precedence, and user groups.

4.2. An important role of any protocol is to present a simplified view of the communication medium (which may be quite complex in reality) to its users. To this end we think it is useful to speak of process-to-process communication, without regard to the actual host level entity (host, TAC, or CCU) within which the process resides. A process, as used here, is that active element in the host level entity that actually controls the transfer of information. Even terminals and files or other I/O media are viewed as communicating through the use of processes. Thus, all network communication is viewed as inter-process communication.

4.3. Except for some network performance characteristics, such as throughput, response time, and reliability, the process-to-process communication is in general not dependent on the medium over which the information is transferred. In fact, one of the functions of the host level protocol is to make the network as transparent as possible. It does this by relying on lower level protocols to perform functions such as network routing, PS-PS level communication and error detection, and network flow control.

4.4. Connection Management

4.4.1. To achieve communication between a pair of processes through a switched network, one of the functions of the host level protocol is to provide the means by which a connection may be established between the processes, controlled during the transfer of the information, and terminated at the completion of the session. The termination may not be complete, but may be a transfer of one end of the connection from one process to another process in the same host entity.

4.4.2. The important point is that the connection management function is implemented in a network-transparent manner, i.e., process level connection management is independent of network level connection management, and, from the network standpoint, the network level operations do not depend upon the host level protocol connection management.

4.4.3. Most processes interact with input or output sources or streams. We imagine that each process may have a number of these input and output streams through which it communicates with other processes. Programmers refer to such streams as channels, logical unit numbers, job file numbers, etc.

4.4.4. Since the TCP may manage several connections concurrently it needs to distinguish among the input and output streams for the various connections. The TCP assigns a Local Connection Name (LCN) to each stream so they may be distinguished at the process-TCP interface.

4.4.5. A similar naming convention is necessary for the streams between processes in the network, with important constraints--the names must have the same syntax at every host, and some names must refer to the same functional process over long periods. To provide such a universal name space for streams the concept of a PORT is introduced. There may be host-specific mappings between ports and streams.

4.4.6. It must be possible for processes to select a port name from either a static or dynamic set. A port name drawn from a static set is intended to be used and reused over a long period and could be based on the user's name or directory number. A port name drawn from a dynamic set is intended to be used for this session only and could be based on the current process identifier or the time of day.

4.4.7. Since port names are selected independently by each operating system, TCP, or user, they may not be unique. To provide for unique names at each TCP, we concatenate a TCP identifier with a port name to create a socket name which will be unique throughout the network. More on user and TCP addressing can be found in Appendix A.

4.4.8. A pair of sockets form a connection which can be used to carry data in either direction (i.e. full duplex). The connection is uniquely identified by the <local socket, foreign socket> address pair, and the same local socket can participate in multiple connections to different foreign sockets.

4.5. Information Transfer Management

4.5.1. The second major function of the host level protocol, possible after the establishment of a connection, is the management of the actual transfer of information. This involves such functions as synchronization, sequencing, delivery acknowledgment, error detection, identification and elimination of duplicates, nondelivery detection and recovery, and process level flow control. This function is also implemented in a network-transparent manner, and should be, to the maximum extent possible, independent of such network constraints as packet size, network routing, network configuration changes, and PS-PS control procedures.

4.5.2. Processes make use of the TCP by handing it letters. The TCP breaks these into segments (called T-segments), if necessary, and then embeds each segment in an S-segment suitable for transmission from the host to its serving PS. The PS's may perform further formatting or other operations to deliver the local packet to the destination host.

4.6. Security, Precedence, and TCC

4.6.1. There are three operational characteristics that the AUTODIN II host level protocols contain in addition to those described in Appendix D of the AUTODIN II Specification: these are the unique DoD common user requirements of security, traffic acceptance categorization (precedence), and closed user groups (TCC).

4.6.2. The security, precedence, and TCC requirements, described in detail in various sections of this document (see for example sections 5. and 9.1.3.), apply to both the connection management and the information transfer management functions of the host level protocol. The host level protocol provides the capability to handle multiple connections of varying security, precedence, and TCC requirements, in the same host entity.

5. DISCUSSION OF S/P/T AND CONNECTIONS

5.1. The AUTODIN II specification indicates that unique security, precedence, and closed user groups problems must be handled by the host-to-host protocol. The PS is responsible for preventing security, precedence, and user group violations. However, the PS can check for violations only in a simplex or unidirectional communication path.

5.2. Certain restrictions concerning security, precedence, and user group are desirable on a full duplex connection. There are restrictions on the usage of connections imposed for administrative reasons, such as restricting all data that flows on a connection to exactly the security level indicated at the time the connection is established. Other restrictions may be derived from implementation constraints or administrative requirements. For example both directions of data flow on a connection must be at the highest precedence level of any data that is sent on that connection, even if the critical data flows only one way. This is necessary because data flow in one direction must be acknowledged by messages transmitted in the other direction. The acknowledgements must travel at the same precedence as the data to prevent a slow down or stopping of data transmission caused by flow control.

5.3. In another instance, a process waiting to receive data from the network at a security level higher than that allowed by the switch will wait forever, not knowing why data has not arrived. (It should be noted that no network traffic is generated until the actual sending of a message over a TCP connection.) At the switch, no information exists that could be used to manage full duplex connections; and in the host, TAC, or CCU no information exists that can be used to monitor security, precedence, and user group (TCC).

5.4. To remedy the problems related to administering S/P/T control at the TCP level, it is recommended that associated with each computer housing a TCP (or any other host-to-host protocol program) there be an authorization table from which the TCP can extract S/P/T information. A more detailed discussion of issues relating to such a table is presented below.

5.5. Security

5.5.1. The TCP should be able to enforce tighter monitoring of security than that provided by the PS. In addition, a user who is allowed to communicate at a specific security level may choose to limit his activity to some lower level. These cannot be monitored by the switch and must therefore be the task of the TCP.

5.5.2. There are currently 16 levels of security of which the lowest is the least classified and highest the most classified. When specifying a security level for his connection a user may choose one of two possibilities: (1) A unique level, indicating he is willing to communicate on this connection only at the specified level or, (2) a ceiling, indicating he is willing to accept any security level not exceeding the one specified.

5.5.3. The strategies of monitoring the security and the process of establishing the security of a connection are described in 9.1.3.2.

5.6. Precedence

5.6.1. Precedence is a means of assigning urgency to the data being sent over the connection. There are 16 levels of precedence currently available to the user, of which the highest is the most urgent. As with security, the precedence is checked in two levels--the PS level and the TCP level. Category I, the highest precedence is the only non-blocking precedence at the PS level, and the only one that can cause preemption in the TCP.

5.6.2. Unlike security, the precedence of a connection must remain constant over the lifetime of the connection. The PS monitors the eligibility of the subscriber to use various precedences while the TCP monitors the consistency of the precedence during the connection lifetime (see 9.1.3.3.).

5.7. Transmission Control Code (TCC)

5.7.1. A TCC defines a closed group of users for whom the network provides a communication medium. Membership in a TCC gives a user the capability to send to and receive from any other member of the TCC. A user must be in at least one TCC, but may belong to several of them. Groups may overlap. TCC's are a mechanism to restrict the access between specific user groups, but they do not restrict the mode of access. Users that are allowed to communicate are assumed to have full access to one another, i.e. to send and receive data as well as control.

5.7.2. Enforcing TCC's is the combined responsibility of the PS and the TCP. Each has a different view of the member entities of a TCC, thus each accesses a different data structure when checking TCC's. The PS accesses the switch directory, which holds a list of TCC's for each subscriber. The TCP accesses an authorization table described below, which holds a list of TCC's for each user.

5.7.3. The packet switch behaves identically in all instances with respect to TCC checking. The switch has knowledge of neither the bidirectional nature of TCP connections nor host users; therefore, it can only restrict the delivery of a message based on the TCC list corresponding to the subscriber. The only function performed in the switch is to examine the TCC of each segment and compare it against the TCC list for that subscriber. For incoming segments, a violation produces a message to the remote switch. For outgoing segments, a violation results in a message to the local SIP.

5.7.4. Additional TCC implementation considerations can be found in 9.1.3.4.

5.8. Authorization Table

5.8.1. The TCP will retrieve all S/P/T information from an authorization table residing in the operating system. This table will not be merely a redundant switch directory, but a table with entries for each process (user on a host, subscriber on a TAC) that exerts more restrictive control than the switch directory. The authorization table entries will hold the following:

5.8.1.1. The range of acceptable security levels for sending and receiving.

5.8.1.2. The maximum precedence level for sending.

5.8.1.3. A list or map of all valid TCC's.

5.8.1.4. Any other information, relevant to the TCP, which is controlled by the operating system (such as the authorization to stop transmission).

5.8.2. A graphical view of the authorization table is shown in Figure 4.

5.8.3. Users that choose to implement a different host-to-host protocol may also need an authorization table to achieve the desired efficiency.

5.8.4. The TCP will retrieve information from the authorization table by calling the operating system indicating the ID of the

AUTHORIZATION TABLE

	Security				Precedence	User Group
	Net To User		User To Net		maximum allowed	list of allowed TCC's
	Max	Min	Max	Min		
User 1						
User 2						
User 3						
⋮						
User n						

Figure 4

process in question. The operating system will map the process ID to the user's authorization table entry and return the entire entry to the TCP.

5.8.5. In the absence of an authorization table accessible by the TCP, the issues raised above cannot be gracefully handled. Below is a summary of the most serious deficiencies encountered in the absence of an authorization table.

5.8.5.1. The authorization table enables the TCP to monitor S/P/T controls more strictly than does the PS. For a TAC subscriber the control exerted by the PS might suffice. However, in the host case, since the PS cannot distinguish between users the S/P/T control applies equally to all users and individual bounds are impossible.

5.8.5.2. The creation of a listen connection, i.e. one which is partially unspecified in terms of foreign socket or S/P/T parameters, may cause the user to listen indefinitely if the S/P/T parameters specified are in violation of his authority. Although a user opening an unspecified connection indicates his intent to wait indefinitely it is a waste of resources if the connection will never become established. The availability of an authorization table eliminates such cases.

5.8.5.3. The PS and all protocols lower than the TCP operate in a simplex, or unidirectional mode. The accessibility of the TCP to the authorization table will allow full duplex monitoring of S/P/T. In the absence of the authorization table S/P/T cannot be monitored in a full duplex manner, weakening the fundamental assumption of a complete full duplex connection from the user's standpoint.

5.8.6. It should be noticed that in reality a user may obtain the contents of its own switch directory entry by successive attempts to send letters with various S/P/T parameters. Hence, the authorization table does not provide the user with any information which he could otherwise not obtain.

6. TCP ENVIRONMENT

6.1. The program that implements the TCP is not a self-contained system. It has to communicate with the processes it serves and with the operating system under which it runs. Part of the TCP's environment is its interface with its neighbors, the SIP on the network side, the user process (THP or other) on the user's side. The interface with the SIP and the user is given in more detail in the next section. A description of the envisioned operating system environment in which the TCP operates is outlined below.

6.2. Operating System Characteristics

6.2.1. Here we describe some operating system characteristics that are important for any implementation of TCP. In no way is it implied that these characteristics are absolutely required of any operating system under which TCP is to be implemented. Those characteristics that are most important to the foundation of the specification are discussed first, followed by some characteristics that would generally facilitate implementation of TCP.

6.2.2. Strong Recommendations

6.2.2.1. Address spaces and data sharing

6.2.2.1.1. In an environment of cooperating processes in which large blocks of data are being passed between the processes, it is often important that the processes be able to pass data without copying it. It is therefore advisable that processes be able to share the physical address space in which the data resides or have some other mechanism for efficient data sharing.

6.2.2.2. Inter-process communication

6.2.2.2.1. The host inter-process communication mechanism should support asynchronous calls such that the user is not blocked waiting for I/O with the network or other I/O device. A more detailed discussion of interprocess communication is presented in section 6.3.

6.2.2.3. Preemption

6.2.2.3.1. Some mechanism must be available to secure resources, both time and space, when high precedence activity is encountered. The operating system should have features in its process management, process scheduling, and space management that permit preemption. Some suggestions that relate to preemption are offered in the next section.

6.2.2.3.2. When in need of resources, the TCP should first attempt to preempt resources it has itself allocated, prior to obtaining further resources from the operating system. In an extremely heavy load preemption within the TCP may be impossible and the operating system's assistance might be needed to carry out preemption. A detailed description of the mechanism of preemption within the TCP is given in section 9.5.4.

6.2.3. Suggestions

6.2.3.1. Process structure

6.2.3.1.1. It is not necessary to define the host, TAC, or CCU modules (SIP, TCP, THP, TC, HSI) or module subtasks as processes of the operating system. A process can be thought of as "a running program"; the current code, resources, and context. While it may not matter if a process is known or managed directly by the operating system, it is conceptually useful to be aware of the hierarchy of processes. In the host, TAC, or CCU, the superior of a process is defined as the element that activates that process (usually some kind of scheduler).

6.2.3.2. Scheduling

6.2.3.2.1. Scheduling is performed at several levels in a multi-process environment and depending upon process structure, can be arbitrarily complex. Of concern here is the scheduling of top level processes (SIP, TCP, THP, TC, HSI) and the pitfalls to be avoided at all levels of scheduling. No mention is made here of host process scheduling; it is assumed that scheduling suggestions are helpful only for TAC'S, MCCU's, and SCCU's. Additional scheduling notes may be found in the discussion of TCP Implementation (Section 8).

6.2.3.2.2. TAC and CCU scheduling is primarily concerned with fairness in function and should be non-preemptive. Scheduling should be done using a round-robin scheme. To insure some level of fairness between top-level processes, processes should limit their activity to some agreed upon maximum. To prevent deadlocks, processes should not block themselves waiting for resources to be freed or provided by other processes at that level.

6.2.3.2.3. In several of the TAC and CCU modules, special handling of high precedence activity is required. Scheduling according to the precedence of an activity should be handled in the respective processes, in a way that does not preempt a task in execution. For example, within the TCP, no event should preempt the running of the from-net segment handler. After the from-net segment handler finishes with the present segment, however, the scheduler gains control and will reexamine the TCP event list for high precedence events before another process is scheduled.

6.2.3.3. Buffer management

6.2.3.3.1. Buffer management in the host, TAC, and CCU is a difficult problem and the "correct" algorithm will evolve only after performance measurements are made. The strategy must address several issues--flow control, deadlock avoidance, and throughput. As with scheduling, it is probably incorrect to handle all buffer allocation uniformly. Where special heuristics are required for solving allocation problems, the allocation should be handled at the process level (e.g. allocating buffers for segmentizing letters in the TCP).

6.2.3.3.2. To facilitate buffer management we have adopted a policy in which the module that requests allocation of space is the one that releases it. Thus buffers allocated to the user will be released by her, buffers allocated to the TCP will be released by the TCP, and buffers allocated to the SIP will be released by the SIP.

6.2.3.3.3. We envision at least two levels of space or buffer management--global management for allocation to the user, TCP, and SIP modules, and local management for allocation within each of the modules. For instance, the TCP space manager (a local manager) acquires space from the global manager and in turn allocates space to its modules, independently of the policy used by the other local managers.

6.2.3.3.4. The global manager is responsible for fairness among the user, TCP, and SIP modules. Since no buffers actually move across interfaces permanently, this fairness is straightforward to implement. The global manager cannot, however, address problems like deadlock prevention since it cannot determine how its allocations are to be used. Only the local managers, especially the user space manager, can attempt to distribute buffers to input and output traffic in such a way as to avoid deadlocks.

6.3. Inter-Process Communication

6.3.1. In implementing the TCP protocol the need for asynchronous communication is very significant. To emphasize this point we will describe all inter-module communication in terms of events (with no distinguishing external characteristics). This is in contrast to earlier documents on host-to-host and process-to-process protocols, which describe inter-module communication in terms of procedure calls.

6.3.2. Process A must be able to signal Process B that event data is ready for it. Process A must regain control immediately following posting of that event with some indication that the event was posted successfully or unsuccessfully.

6.3.3. Posting events is an operation to be performed by the operating system (or some event sending mechanism running under the operating system). It enables the operating system to verify the authority of the event sender to communicate with the event receiver. It also informs the sender if an attempt is made to send an event to a non-existing process or to a process that has just crashed.

6.3.4. To facilitate event handling a process ID must be associated with each process. The process ID is the means by which the sender specifies the destination of the event. The operating system will attach the sender's ID to each event thereby enabling the destination to distinguish among various sources he might be communicating with, and at the same time, prevent the sending process from masquerading as another.

6.3.5. Associated with each TCP event are at least the two fields: "T-seg I-D" and "Reason". All events are handled by the operating system in the same priority level. Each process (TCP or SIP) should keep a list of events it has read but not yet serviced completely. Before starting on the highest priority listed event the process should read and enter into the list any newly received events.

6.3.6. Event data should be passed by value, i.e. copied from the sender's address space. The data should be copied into a sharable memory for fast access by the receiving process. It should be fixed in length and small so that allocation for event data is efficient. The event data could be thought of as being placed on the receiver's event list, which should be dynamic in length and never blocked.

7. TCP INTERFACES

7.1. USER-TCP Interface

7.1.1. The TCP can be viewed by the user as an efficient post office that handles his (certified) letters. The user hands the TCP letters addressed to another user (or process), and expects the TCP to arrange for the delivery of the letter to the addressee, notifying him upon delivery. The user is not assumed to have any knowledge of the mechanism employed by his TCP to carry out the delivery of his letter. Consequently, he can communicate with others using only a set of primitive instructions such as SEND, CLOSE, etc.

7.1.2. Users communicate over the network via connections. Communication between the user and the TCP is performed in an asynchronous manner, i.e., the user does not have to wait for the results of his request, but rather will be notified of successful completion or an error condition.

7.1.3. To communicate with a remote process, the user's process must understand the concepts of a connection, the addressing mechanism, and the "language" that his TCP understands. These are explained below.

7.1.4. Addressing

7.1.4.1. We assume that each process may have a number of ports through which it communicates with the ports of remote processes. It is possible to specify a remote socket only partially by not specifying (setting to zero) the port identifier, or both the TCP and port identifiers. A socket of all zero is called unspecified. The role of unspecified sockets is to provide a sort of "general delivery" facility (useful for logger type processes with well known sockets or some other listening processes). Appendix A gives more detail about user and TCP addressing.

7.1.4.2. Once the connection is specified in the OPEN command (see section 7.1.6.3.), the TCP supplies a short Local Connection Name (LCN) by which the user refers to the connection in subsequent commands. There are limits to the degree of unspecificity of socket identifiers. Local sockets must be fully specified, although the foreign socket may be fully or partly unspecified. Arriving segments must have fully specified sockets.

7.1.4.3. Each connection is associated with exactly one process, and any attempt to reference that connection by another process will be signalled as an error by the TCP. This prevents stealing data from or inserting data into another process' data stream.

7.1.4.4. To prevent TCP masquerading, a TCP does not know its own identity for specifying the local socket, as TCP identifiers will be supplied by the operating system under which it runs.

7.1.5. Connections

7.1.5.1. To facilitate communication between a pair of processes through the network, the TCP provides the means by which a connection between the processes is established, controlled during the transfer of the information, and terminated at the completion of the session.

7.1.5.2. A pair of sockets form a full duplex connection capable of carrying data in either direction. A connection is uniquely identified by the <local socket, foreign socket> address pair. The same local socket can however, participate in multiple connections to different foreign sockets.

7.1.5.3. Associated with each connection are S/P/T values that are monitored by the TCP over the lifetime of the connection.

7.1.6. Primitives for USER-TCP Communication

7.1.6.1. In the following paragraphs we shall outline the possible events that may cross the user-TCP boundary. These are the commands or primitives with which a user's process may communicate with any other user (or process) across the network. Event sending does not support a call/return relationship between modules. Hence, this relationship must be achieved by convention. In the description of the primitives available to the user, it is important to realize that a pair of events are involved, the call event and the return event. The return event is sent when some significant activity has occurred. (See Appendix B.)

7.1.6.2. Two special events, ACK and NACK, are used to notify the sender of an event that the event parameters are (are not) acceptable and that the action associated with the event will (will not) be attempted.

7.1.6.3. OPEN CONNECTION

7.1.6.3.1. This is the first action the user undertakes to initiate communication on a connection. The user's request is recorded by his TCP but no network traffic is generated by this event. The first SEND or INTERRUPT by the local user or the foreign user will cause the TCP to synchronize the connection.

7.1.6.3.2. In the OPEN request the user may specify the foreign socket with which he wishes to communicate and the S/P/T parameters for the connection. If the foreign socket or the S/P/T (or a combination of these parameters) are not specified, then this constitutes a LISTENing local socket which can accept a connection attempt from any foreign socket that matches the parameters specified.

7.1.6.3.3. The S/P/T parameters may contain the send security level (or range), the intended send precedence, the minimum acceptable receive precedence, and the TCC for the connection. A user is permitted not to specify any of these, indicating his willingness to accept connection attempts with any values for the parameters he did not specify (provided he is eligible to use them).

7.1.6.3.4. If the specified connection is already OPEN or the S/P/T parameters are in violation of his authority, an error is returned.

7.1.6.3.5. If the OPEN is successfully recorded by the TCP, a local connection name (LCN) will be returned to the user. The local connection name can then be used as a short-hand name for the connection defined by the <local socket, foreign socket> pair.

7.1.6.3.6. The local TCP must be aware of the identity of the processes it serves and will check the authority of the process to use the socket specified as well as the validity of the S/P/T.

7.1.6.3.7. Section 9.1.1. contains more implementation details about opening connections.

7.1.6.4. SEND LETTER

7.1.6.4.1. This call causes the data contained in an indicated user's buffer to be sent on the indicated connection. If the connection has not been opened, or the calling process is not authorized to use this connection, an error event is returned.

7.1.6.4.2. The user has to specify, by means of a specially designated flag, whether this data is the End-of-Letter. If the EOL flag is not set, subsequent SEND's will appear as part of the same letter. This extended letter facility should be used intelligently because some TCP's may delay processing packets until an entire letter is received.

7.1.6.4.3. The security level of the letter accompanies each SEND event. This accommodates changing of the security on each letter, provided the connection was opened with a security ceiling specified.

7.1.6.4.4. If no foreign socket was specified in the OPEN, but the connection is established (e.g. because a LISTENing connection has become bound due to the arrival of a foreign letter for the local port) then the designated letter is sent to the bound foreign socket.

7.1.6.4.5. If a SEND is attempted before the foreign socket becomes specified, or after a CLOSE is requested by either party, an error will be returned.

7.1.6.5. RECEIVE LETTER

7.1.6.5.1. This command offers a receiving buffer for this connection. If no OPEN precedes this command or the calling process is not the owner of this connection, an error is returned.

7.1.6.5.2. If insufficient buffer space is given to reassemble a complete letter, the buffer will be filled with as much data as it can hold, and an indication that the buffer holds a partial letter will be given.

7.1.6.5.3. The remaining parts of a partly delivered letter will be placed in buffers as they are made available via successive RECEIVES. If a number of RECEIVES are outstanding, they may be filled with parts of a single long letter or with at most one letter each. An EOL flag associated with each RECEIVE return event will indicate if the contents include the end of the letter.

7.1.6.5.4. The user has to specify the mode in which he wishes to have his buffer returned. In the FULL mode the buffer will be returned to the user as soon as it is filled, or a complete letter has been assembled. In the PARTIAL mode the TCP will return the buffer to the user as soon as it has any data available, i.e. when the buffer is non-empty and one of the following has occurred: no more data in the TCP awaits transfer to the user, the buffer is full, or a complete letter has been assembled.

7.1.6.6. RETRACT RECEIVE BUFFERS

7.1.6.6.1. Retracting receive buffers may be used to recover resources or to stop the flow of data from the TCP to the user. While the primitive may find some limited utility for recovering previous offered receive buffers, the primary use is seen to be in user-to-user flow stoppage. The available receive buffer space is used by the TCP in its flow control strategy in such a way that retracting buffers causes the local TCP to stop data flow from the remote TCP (and the remote user).

7.1.6.7. CLOSE CONNECTION

7.1.6.7.1. This command causes the connection to be closed. If the connection is not open or the calling process is not authorized to use this connection, an error is returned. Any unfilled receive buffers will be returned to the user with event codes indicating they were aborted.

7.1.6.7.2. Two types of CLOSE may be used: a graceful close (Deferred) and a flushing close (Immediate). In the deferred close the connection will be closed only after all pending letters have been fully acknowledged, whereas in the immediate close pending letters will be discarded and the connection closed.

7.1.6.7.3. The connection may be closed at any time by the user, or by the TCP in response to various error conditions (e.g. remote close executed, transmission timeout exceeded, destination inaccessible).

7.1.6.7.4. Because closing a connection requires communication with the foreign TCP, connections may remain in the closing state for a short time. Attempts to reopen the connection before the TCP replies to the CLOSE command will result in an error message to the user.

7.1.6.7.5. More detailed implementation suggestions can be found in section 9.1.4.

7.1.6.8. INTERRUPT

7.1.6.8.1. When this primitive is called by the user, a special control signal is sent to the destination indicating an interrupt condition. This facility can be used to simulate "break" signals from terminals, or error or completion codes from I/O devices, for example. The semantics of this signal to the receiving process are unspecified.

7.1.6.8.2. Two types of INTERRUPT may be used: a graceful interrupt (Deferred) and a flushing interrupt (Immediate). In the deferred interrupt the signal will be associated with the end of the last letter sent, whereas in the immediate interrupt pending letters will be discarded at both the local and foreign TCP's. The receiving TCP will signal the interrupt to the receiving process upon receipt.

7.1.6.8.3. If the connection is not open or the calling process is not authorized to use this connection, an error is returned.

7.1.6.9. FLUSH

7.1.6.9.1. Under certain circumstances it may be desirable to flush the pipe between the sender and receiver without sending the out-of-band FL-INT signal. The FLUSH primitive instructs the local TCP to flush as much of the outgoing data as it can by not sending anything it has recently segmentized and by not retransmitting any unacknowledged segment.

7.1.6.9.2. The local TCP will send a FL control to the remote TCP, informing it to return all outstanding receive buffers to its user with an accompanying message informing her of the flush.

7.1.6.9.3. If the connection is not open or the calling process is not authorized to use this connection, an error is returned.

7.1.6.10. STATUS

7.1.6.10.1. This command returns a data block containing connection-related information. Depending on the state of the connection, some of the information may not be available or meaningful.

7.1.6.10.2. The information returned by the STATUS primitive is mostly site dependent and will not be described in detail. It must include the send buffer address for which the last acknowledgements have arrived from the remote user and the number of octets successfully delivered from that buffer to the remote user. This is necessary for synchronizing of data transfer at the user level.

7.1.6.10.3. If the calling process is not authorized to use this connection, an error is returned. This prevents unauthorized processes from gaining information about a connection.

7.1.6.11. MOVE CONNECTION

7.1.6.11.1. We have stated before that a connection is owned by the process that first opened it. It is useful, however, to be able to transfer ownership of a connection from one process to another. This is made possible by the MOVE primitive, issued by the original owner of the connection.

7.1.6.11.2. The important point behind the ownership transfer process is that the TCP has to verify the authority of the new owner to own the specified connection in terms of its S/P/T parameters.

7.1.6.11.3. If the new owner is not authorized to use the connection the request is rejected, i.e. the old owner is notified and ownership is not transferred.

7.1.6.11.4. It should be noted that at all times the connection is owned by some process (the old or new user), and that the transfer cannot cause any malfunction in the TCP even if the new owner has crashed or fails to use the connection. More detail is provided in 9.1.

7.1.6.12. GENERAL

7.1.6.12.1. In the user-to-TCP direction, this event provides a path for user-to-packet switch messages. Currently, only one such message is specified--"host going down."

7.1.6.12.2. In the TCP-to-user direction, this event is the means of providing the user with information not directly related to a previous action or event initiated by her. (Note it does not follow any call/return conventions.) This event will be used to notify the user of modules (hardware and software) that are going inoperable, error messages of local or remote origin, and arrival of letters for which no RECEIVE has been issued by the user.

7.1.6.13. STOP TRANSMISSION

7.1.6.13.1. A privileged user issues this command to request that the PS discard segments from the designated source. The primary use of this command is to prevent a remote subscriber from sending segments that are either erroneous or bothersome (e.g. uses valuable bandwidth with no useful results).

7.1.6.13.2. The TCP does not keep record of the sources from which transmission was stopped. Thus, the user should keep that information to avoid connection attempts to sources from which transmission is stopped.

7.1.6.13.3. If the connection is not open or the calling process is not authorized to use this connection, an error is returned.

7.1.6.13.4. The stop transmission command is an extremely powerful one, especially in the host case; if issued by one user it will stop transmission from the designated address to ALL users of the local host. Also, if the subscriber from which transmission is stopped is a host then transmission is stopped from ALL users of that host.

7.1.6.13.5. Consequently, the stop transmission command will be reserved for privileged users only. To enable the TCP to distinguish between a regular and a privileged user it is further recommended that this capability be recorded in the authorization table, or alternately passed to the TCP when it attempts to retrieve authorization information about a process.

7.1.6.13.6. In addition to the stop transmission capability from a designated subscriber, the TCP supports commands to stop transmission from a designated TCC.

7.1.6.14. RESUME TRANSMISSION

7.1.6.14.1. This command enables the privileged user to resume transmission from the designated source. A resume transmission from a source which has not been stopped before is considered a no-op and the user will not be notified. A "resume all" is used as a reset command that resumes transmission from all sources.

7.1.6.14.2. As in the case of the stop transmission command, only privileged users (those who may stop transmission) will be able to issue this command.

7.1.6.14.3. If the connection is not open or the calling process is not authorized to use this connection, an error is returned.

7.2. TCP-SIP Interface

7.2.1. The SIP serves as the interface between the TCP and the network. Its main interfacing functions are to pass controls from the TCP to the local PS and vice versa, and to transmit data from the local TCP to a remote TCP.

7.2.2. Communication between the TCP and the SIP follows the event model presented before for the user-TCP interface. The TCP will send events to the SIP which, in return, will take the appropriate action. As far as the TCP is concerned the SIP should be able to process the set of events described below and follow some configuration dependent addressing conventions. To retain program modularity, the TCP does not assume any knowledge of the mechanism employed by the SIP to perform its tasks.

7.2.3. SIP Addressing Responsibilities

7.2.3.1. The SIP must know how to route segments to/from its neighbors. Depending on the configuration, the SIP may be passing segments to/from a TCP which is serving one subscriber (the host) or, many subscribers (the TAC).

7.2.3.2. In the host case the addressing is straightforward. The host has many users, but the composite set of users comprises only one subscriber. No multiplexing is needed by the SIP to support subscriber addressing. The TCP will provide user multiplexing.

7.2.3.3. In the TAC case each user is a distinct subscriber, TAC – LCM INTERFACE

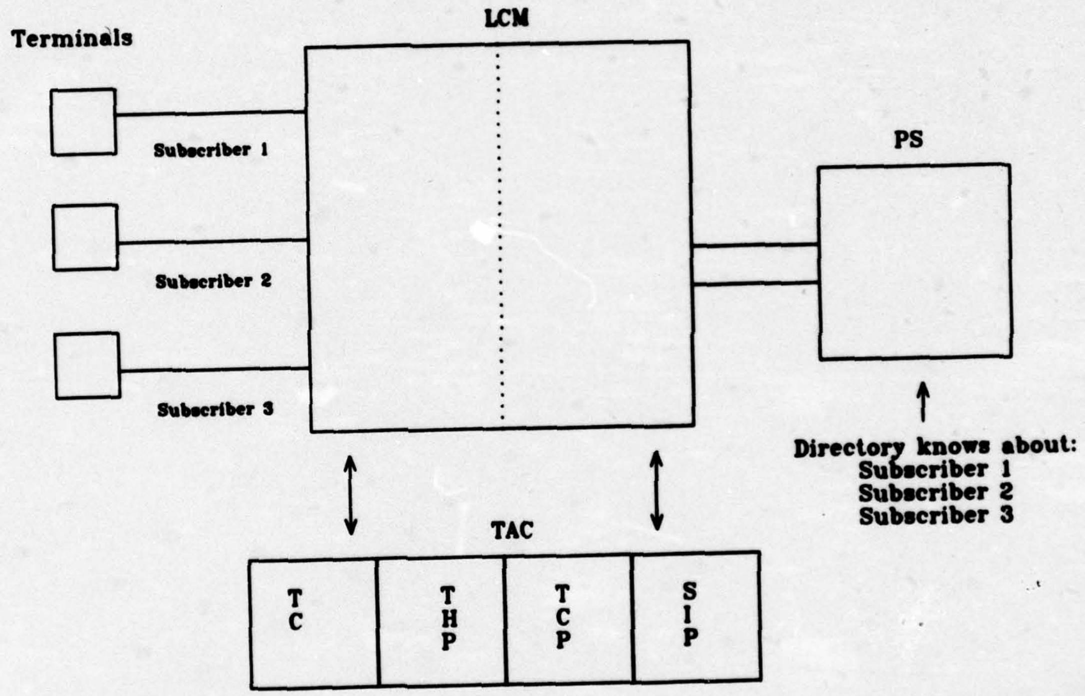


Figure 5

however, the TAC itself is also a subscriber. Messages destined for the user are first passed to the TAC and then forwarded to the user. The SIP-PS message leader, the Binary Segment Leader, is not sufficient to handle this additional multiplexing requirement. The SIP will add a prefix containing the source terminal's subscriber ID to the Binary Segment Leader so that the PS will know the real source of the segment. In the opposite direction, it is assumed that the PS or LCM will include a prefix containing the subscriber ID of the destination terminal on segments delivered to the SIP of a TAC, so that the TAC can route the segment to the terminal of the subscriber. Figure 5 illustrates this TAC-LCM interface.

7.2.4. Primitives for TCP-SIP Communication

7.2.4.1. In the following paragraphs we shall describe the set of events by which the TCP and the SIP communicate. All such events contain at least three fields: Operation code, subscriber ID, and T-segment ID. The subscriber ID enables the TCP and SIP to identify the subscriber to which the event relates, and the T-Segment ID makes it possible to associate an event with a specific segment. Hence the TCP and SIP can converse about a specific segment of a specific user. Events with unspecified subscriber ID or unspecified T-segment ID refer to all the subscribers connected to a TCP or all segments respectively.

7.2.4.2. DATA

7.2.4.2.1. This event may cross the TCP-SIP boundary in either direction. The TCP is notified by this event of the arrival of a segment addressed to it. The SIP is notified by the TCP via this event that a segment is now ready for transmission on the network.

7.2.4.3. ACK / Negative ACK

7.2.4.3.1. The TCP and SIP use the ACK event to acknowledge that the indicated segment was accepted and will be processed. Via Negative ACK event the TCP and SIP notify each other that the indicated segment cannot be processed because of incorrect or invalid parameters.

7.2.4.4. STOP / RESUME Transmission

7.2.4.4.1. Via this event the TCP conveys to the SIP a user's request to stop or resume transmission. Although not currently specified in the AUTODIN II specification, provisions are made at the TCP level to support stop and resume transmission based on user groups.

7.2.4.4.2. A "RESUME ALL" is a reset command and is used by the TCP to clear all transmission paths. An attempt to "STOP ALL" is an error and shall be rejected by the SIP.

7.2.4.5. Going Inoperable

7.2.4.5.1. This event is the means by which the TCP reports to the PS (via the SIP) of any hardware module that is going inoperable. Each such event will specify an estimated time for how long the relevant module will be inoperable and the reason for it.

7.2.4.6. ERROR

7.2.4.6.1. This event is the means by which the SIP notifies the TCP of errors reported by the PS. Errors may relate to a previously sent segment or to a change in network status, such as rejection by the (remote or local) PS because of S/P/T or address violation, or reports of undelivered segments. A description of the possible errors is given in Appendix B.

8. TCP IMPLEMENTATION

8.1. Introduction

8.1.1. Any particular TCP implementation could be viewed in a number of ways. It could be modelled as an independent process, servicing many user processes. It could be viewed as a set of re-entrant library routines which share a common interface to the local SIP, and common buffer storage. It could even be viewed as a set of processes, some handling the user, some the input of segments from the network, and some the output of segments to the network.

8.1.2. The implementation model selected for this document is closest to the latter. The TCP is one superior process (external event handler), which (1) interacts with its neighboring modules, the user process or SIP, using event posting, and (2) schedules a set of four inferior processes--segmentizer, to-net segment handler, from-net segment handler, and reassembler (see Figure 6). The TCP is modelled as one instance, which handles the multiplexing of all TCP connections.

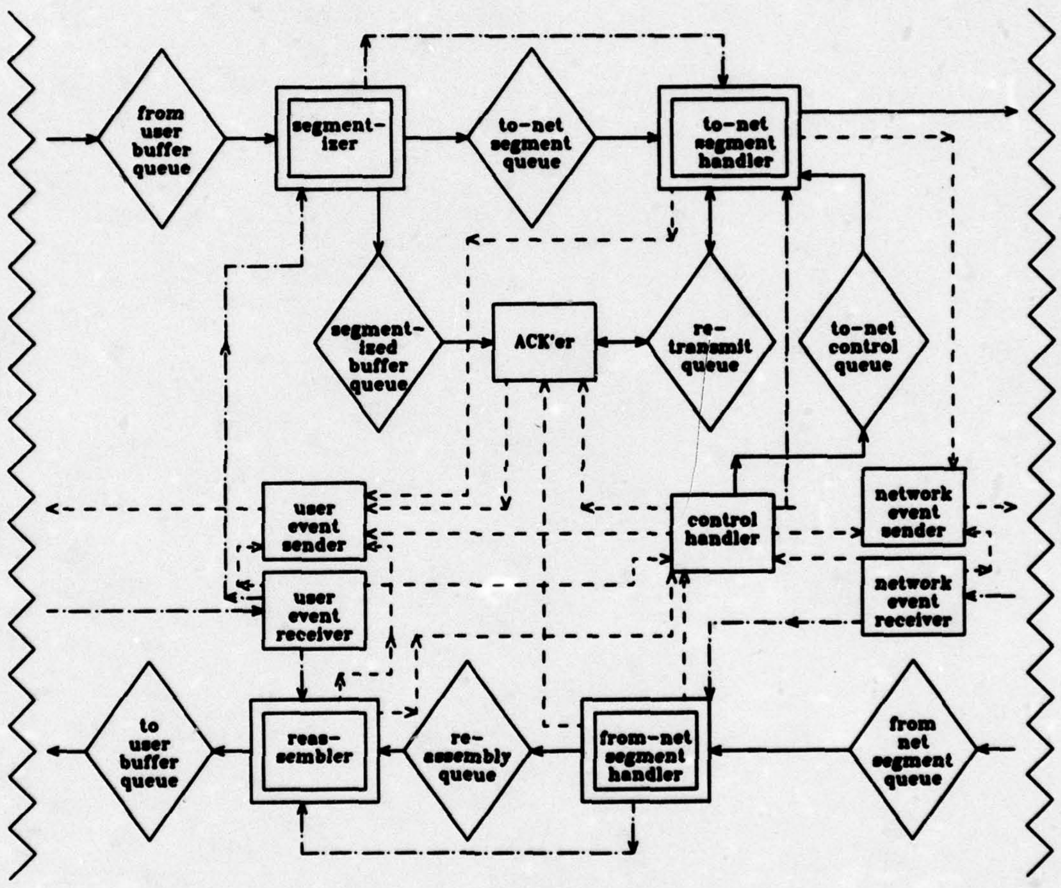
8.1.3. The intra-TCP processes access several data structures, most of which are shared. Communication between intra-TCP processes is achieved through an intra-TCP signal board. Segments and letters move through the TCP on shared queues. The states of connections are maintained in shared Transmission Control Blocks (TCB's). All of these data structures are discussed later in section 8.2.

8.1.4. A set of subroutines is shared among the processes. They are invoked by call, in contrast to the processes which are scheduled. The total set of processes and subroutines are the functional modules of the TCP and are described in section 8.3.

8.1.5. Figure 6 shows the relationships among the functional modules of the TCP and the segment and letter queues. Segment and letter flow is shown in solid lines, signal-type control in dot-dashed lines, and subroutine call control in dashed lines. To aid in understanding the relationships, a brief scenario of typical operations is given below.

8.1.6. The user sends a letter to the TCP with a SEND event. When the TCP is allowed to run, the external event handler (not shown) moves that event to an internal ordered event list. It then calls the user event receiver to process the event data,

TCP MODULES AND QUEUES



LEGEND

Buffer movement	———
Synchronous calls	- - - -
Asynchronous calls	-·-·-

Figure 6

which includes the address of the user's letter. The user event receiver puts the address of the user's letter buffer on the from-user buffer queue and signals the segmentizer by setting a bit in the intra-TCP signal board (which is the intra-TCP signalling mechanism). The external event handler then calls the network event receiver to process events from the SIP. Finally it calls the TCP scheduler (not shown), which schedules the four inferior processes according to the priority of the new events received and the signal board entries.

8.1.7. The segmentizer, when scheduled by the TCP scheduler, checks each TCB on its two TCB chains for work, high precedence chain first. The segmentizer checks first for old incomplete segmentizing work and then for new work, i.e. new from-user buffer queue entries. For old work there is a pointer to the location from which segmentizing should resume. Segmentizing requires new buffers and thus the TCP space manager (not shown) must be called to acquire a buffer. As a segment is filled, it is placed on the to-net segment queue and the to-net segment handler is signalled. Once a full letter is segmentized, the letter address is saved on the segmentized buffer queue in case resegmentizing is necessary later.

8.1.8. The *to-net segment handler* is scheduled by the TCP scheduler. It examines the signal board for work. In this example, the board will show at least new to-net segment work. Thus, the segment handler will check each TCB for nonempty to-net segment queues. It will remove a new segment from the to-net segment queue, combine it with controls from the to-net control queue or ACKs that need to be sent, transmit the segment by sending a DATA event to the SIP through a call to the network event sender, and move the segment address to the retransmit queue.

8.1.9. The to-net segment handler could also have been signalled due to a retransmit timeout. In this case, depending on the implementation of timers, the handler would choose the timed-out segment from a retransmit queue and send it through a DATA event to the SIP.

8.1.10. Traffic from the network arrives in SIP buffers. The SIP notifies the TCP of such arrivals through DATA events. These events are handled initially by the external event handler. The network event receiver puts the address of the SIP S-segment buffer on the from-net segment queue and signals the from-net segment handler.

8.1.11. The from-net segment handler is scheduled by the TCP scheduler and locates its work much like the segmentizer and to-net segment handler. The from-net segment queue, however, is a queue that is global to all TCP connections. The handler removes an S-segment from the queue, checks and strips the Binary Segment Leader, interprets the control and ACK fields, and moves any text to the reassembly queue of the destination connection. Controls are passed to the control handler, and ACK's are passed to the ack'er.

8.1.12. The control handler is called by many of the functional modules and may generate TCP controls by enqueueing them on the to-net control queue, generate SIP control S-segments by calling the network event sender, or send a message to the user by calling the user event sender.

8.1.13. The ack'er removes the acknowledged segments from the retransmit queue. If an end-of-letter flag is present in the segment that is acknowledged, it also removes the letter buffer from the segmentized buffer queue and returns the user buffer to the user through a TCP-to-user SEND event (return).

8.1.14. The reassembler process is scheduled by the TCP scheduler. It locates the work by following the TCB chains, high precedence chain first. A nonempty reassembly queue represents work; old work is marked by a pointer to the last data unit copied into the user buffer and a pointer to the last octet (eight bit unit) reassembled from the reassembly queue.

8.1.15. The reassembler copies octets to the next available octet in the next available buffer on the to-user buffer queue. (The to-user buffer queue was populated by RECEIVE events from the user to the TCP, handled by the external event handler and user event receiver.) As each octet is copied to the user's buffer, the receive left window edge is updated to reflect such receipt. When the to-net segment handler runs again, it will send an ACK for everything up to the left window edge. (Windows are used in the flow control mechanism and are explained in sections 9.4. and 9.4.8.) When the reassembler copies an end-of-letter flag or fills the user buffer, it returns the buffer to the user by sending a TCP-to-user RECEIVE event (return) through a call to the user event sender.

8.2. TCP Data Structures

8.2.1. Understanding the data structures of a program is important to the understanding of its functional operation. In this section we present the data structures that the TCP accesses.

8.2.2. Segment Leaders and Headers

8.2.2.1. The Binary Segment Leader (BSL) is a header used for SIP-PS communication. It provides addresses for routing, S/P/T information, and control commands. A T-segment plus a BSL forms a S-segment. The BSL is added to T-segments in the TCP so that buffers can be passed by address from the TCP to the SIP and eventually to the Line Control Module without copying the contents into a new buffer in the SIP.

8.2.2.2. Some fields of the BSL of outgoing segments are actually filled by the TCP, specifically the S/P/T fields. A detailed description of the BSL can be found in Appendix C.

8.2.2.3. To support the TCP-to-TCP protocol a header, the T-segment header, is attached to the user's text by the sending TCP. This header includes routing, flow control, acknowledgement, and control information, and accompanies the user's text to the destination. The header is invisible to and independent of the lower level protocols on which the local TCP depends to deliver the T-segment to the foreign TCP. A detailed description of the T-segment header can be found in Appendix D.

8.2.3. Inter-module Communication Support

8.2.3.1. External Event List

8.2.3.1.1. The TCP's external event list is a data structure that provides a location for the delivery of events and event data for the event sending mechanism. It is an unordered list, each element of which holds the event sender's process ID and the address of a block of event data. The operating system provides primitives that allow a process both to create its own empty event list and to send events to other processes. This data structure may be supported in many ways and is used in this document to model the interprocess communication mechanism used for user-TCP and SIP-TCP communication.

8.2.3.2. Ordered Event List

8.2.3.2.1. The TCP's external event list is a composite list of all pending events from the user or SIP. The TCP internal modules require that these events be separated by source, that the high precedence events are the first to be serviced, and that events at the same precedence are handled first-come-first-served. The ordered event list is populated by the external event receiver and entries are removed by the user and network event receivers.

8.2.3.3. Intra-TCP Signal Board

8.2.3.3.1. The TCP signal board provides an inter-process signalling capability for the intra-TCP processes. An entry on the board is a single bit, settable by any TCP functional module and resettable by the process that it signals. The signalling of processes is an important efficiency measure. It aids the TCP scheduler and the process in determining what work, if any, is pending for the process. Since each process does work for all TCP connections, it would be highly inefficient searching all the TCB's to determine that there is no work pending.

8.2.3.3.2. The following signal board entries and their destinations support process signalling:

from-user buffer queue work (Segmentizer)

to-net segment queue work (To-net segment handler)

to-net control queue work (To-net segment handler)

timeout (To-net segment handler)

ack to send (To-net segment handler)

from-net segment queue work (From-net segment handler)

reassembly queue work (Reassembler)

new user buffers available (Reassembler)

8.2.4. TCB

8.2.4.1. A shared data structure is required to handle bookkeeping on a per connection basis. The data structure should contain enough information to determine the complete state of a connection, locate any segment associated with a connection, and process any control occurring on or for that connection. The data structure that performs these functions in the TCP is called the Transmission Control Block (TCB) which is created and maintained for the lifetime of a given connection. Details on the contents of the TCB are given in Appendix E.

8.2.5. Queue Descriptions

8.2.5.1. For our purposes, a queue is an ordered data structure whose elements are syntactically similar. No assumption is implied about how the elements are ordered nor how adds, reads, and deletes are performed.

8.2.5.2. Queue elements usually include addresses of segment or letter buffers. These buffer addresses are moved between queues; the buffers themselves are never copied. A buffer must exist on one queue only. Nothing is said in this section about the source of the buffer space; however, it is important that buffers passed across the TCP boundaries are addressable by both the TCP and the neighboring module (SIP, THP, etc.).

8.2.5.3. TCP queues hold pending work for the processes. The queues are the mechanism by which processes pass segments to one another, without requiring the immediate attention of the destination process. The queues are accessible to each functional module as described below under Access.

8.2.5.4. Where the structure of the queue or the contents of its entries are important they are described.

8.2.5.5. From-user buffer queue

8.2.5.5.1. The from-user buffer queue is populated by the user event receiver, in response to SENDs by the user, and by the space manager during preemption. The elements of this queue include addresses of letter buffers that belong to the user process and an initial sequence number and size of the letter in octets. The latter items are necessary for the resegmentizing of preempted letters so that resegmentized octets are assigned their original sequence numbers. The queue allows for flow control by the TCP and the queuing of multiple letters by the user process. The user process can have multiple outstanding letters, but must be willing to sacrifice the buffer space for the letters until they have been acknowledged by the remote TCP.

8.2.5.5.2. Queue and element structure

There is one from-user buffer queue per TCB and the head and tail of the queue are stored in the TCB.

Each element must include the user buffer address, the sequence number of the first octet (if previously segmentized), and a letter size in octets.

8.2.5.5.3. Access

ADD:

external event handler (user SEND)
space manager (from segmentized buffer queue)

REMOVE:

segmentizer (finished segmentizing, move to segmentized buffer queue)
cleanup

8.2.5.6. Segmentized buffer queue

8.2.5.6.1. The segmentized buffer queue holds original letter buffers that have already been segmentized, but have not been acknowledged by the remote TCP. The queue is necessary for the case in which a letter is segmentized and a segment of that letter is preempted; the letter buffer can be used to recreate the segment(s). When this happens the buffer is moved to the from-user buffer queue for resegmentizing. Resegmentizing may cause duplicate segments but they are detected and harmlessly acknowledged by the receiving TCP.

8.2.5.6.2. When an ACK is received for the last segment of a letter (the EOL bit was set in the segment on the retransmit queue), the ack'er removes the letter buffer from the segmentized buffer queue and signals the user.

8.2.5.6.3. An extra benefit of the queue is that the letter buffer address is available to be used as a handle for the letter. Thus no other special handle on the letter is necessary when the TCP sends events about that letter to the user process.

8.2.5.6.4. Queue and element structure

There is one segmentized buffer queue per TCB and the head and tail of the queue are stored in the TCB.

The elements of this queue should include the first segment sequence number and the combined length of all the segments created from the letter. The queue should be ordered by sequence number.

During preemption, the space manager can locate the letter corresponding to the sequence number of the segment preempted.

8.2.5.6.5. Access

ADD:

segmentizer (from from-user buffer queue)

REMOVE:

ack'er (all segments acknowledged)
cleanup or space manager

8.2.5.7. To-net segment queue

8.2.5.7.1. The to-net segment queue holds segments to be transmitted to the network (that have never been sent before). The queue will hold data, but not solo ACK segments or control-only segments. The queue is associated with one TCB only and thus can be handled first-in-first-out.

8.2.5.7.2. ACKs are added to the segments just prior to transmission and reflect the current send left window edge.

8.2.5.7.3. Segments are moved from this queue to the retransmit queue after they have been successfully transmitted by the network interface module (SIP). Once a segment leaves this queue, it is never reentered, except through complete resegmentizing in the event of preemption.

8.2.5.7.4. Queue and queue element structure

There is one to-net segment queue per TCB and the head and tail of the queue are stored in the TCB.

8.2.5.7.5. Access

ADD:

segmentizer (from the from-user buffer queue)

REMOVE:

to-net segment handler (move to retransmit queue)
cleanup (flushing)
space manager (preemption)

8.2.5.8. Retransmit queue

8.2.5.8.1. The retransmit queue holds segments that cover sequence numbers that have already been sent. The queue is the source for segment retransmissions, which are necessary if a segment has not been acknowledged within some period of time (usually a little greater than the round-trip time for a network segment).

8.2.5.8.2. Two features about the TCP receiving discipline are important in discussing the retransmission queue--acknowledging of partial segments and shrinking window sizes for flow control. A sending TCP may receive an ACK for part of a previously sent segment, at which time it must determine whether to repackage the remaining part of the segment or to simply retransmit the entire segment. When a receiving TCP shrinks its window, in hopes of slowing transmission from the sender, the sender may respond in exactly the same ways--repackage or send the entire segment.

8.2.5.8.3. To summarize, the operations on the retransmit queue are the following:

Remove or repackage an acknowledged or partially acknowledged segment.

Add a segment from the to-net segment queue.

Find and retransmit a segment.

Repackage all segments for a connection due to window shrinkage.

8.2.5.8.4. Queue and element structure

There is one retransmit queue per TCB and the head and tail of the queue are stored in the TCB.

Retransmit queue elements should be ordered to facilitate removal of an acknowledged segment and the location of segments to be retransmitted. By storing the segment sequence number, segment length, and timeout data in each queue element, and ordering the queue by sequence number, this can be achieved. (Note: the segment sequence number and segment length can be found in the TCP header, but it may be more efficient to address these without the indirection.)

For normal operation, ordering by sequence number should be equivalent to ordering by time-to-retransmit. Even during heavy load, segment acknowledgement should occur more frequently than retransmission, so sequence number ordering still seems superior. Only during very abnormal conditions will retransmissions exceed segment acknowledgements.

Insertions from the to-net segment queue will normally fall at the end (tail) of the queue. Retransmissions do not require reordering of the queue; only an update of the timeout data is necessary. Deletions require some attention, depending on how the elements are addressed, but they should most frequently be from the top (head) of the queue. Removal and reinsertion of repackaged segments should occur at the head.

8.2.5.8.5. Access

ADD:

to-net segment handler (segments just sent)
ack'er (partially acknowledged segments or window shrinkage)

REMOVE:

ack'er (acknowledged segments, partially acknowledged segments, or window shrinkage)
cleanup and space manager

READ:

To-net segment handler (retransmit on timeout)

8.2.5.9. To-net control queue

8.2.5.9.1. The to-net control queue is shared between the control handler and the to-net segment handler. When the control handler deems it necessary to send a control, it adds a control to this queue and signals the to-net segment handler. The to-net segment handler must search the queue exhaustively to find high priority controls. It uses the FIFO nature of the queue to determine order of arrival of the controls.

8.2.5.9.2. Queue and element structure

There is one to-net control queue per TCB and the head and tail of the queue are stored in the TCB.

The queue is FIFO so that the sequence of arrival can be determined. Each element must have a control type and optional data item. The data item could be a sequence number to which the control refers in the case of some TCP-to-TCP Error controls, the sequence number that should be assigned to this control in the case when controls are part of segments to be resegmentized, or arguments for TCP-to-SIP controls in the case of STOP TRANSMISSION or something going down.

8.2.5.9.3. Access

ADD:

control handler (from the following sources)
- from-net segment handler (control)
- reassembler (ack)
- external event handler (interrupt, close, etc.).

REMOVE:

to-net segment handler

8.2.5.10. From-net segment queue

8.2.5.10.1. The from-net segment queue is populated by the from-net event receiver in response to a DATA event received from the SIP. The SIP has no knowledge of TCP connections; thus, there is only one instance of the queue on which all segments for all TCP connections are placed.

8.2.5.10.2. Data segments are moved to the reassembly queue of the appropriate TCB for ordering. Control segments are passed to the control handler and ACKs to the ack'er.

8.2.5.10.3. Queue and element structure

There are two instances of this queue, one for Category I, and the other for lower precedence traffic. This is in contrast to most other queues which have an instance per connection. The elements of this queue are S-segments. Each queue is ordered FIFO.

8.2.5.10.4. Access

ADD:

from-net event receiver

REMOVE:

from-net segment handler (interprets and moves to reassembly queue, if necessary)
cleanup

8.2.5.11. Reassembly queue

8.2.5.11.1. The reassembly queue serves as an ordering mechanism and temporary holding queue from which to build letters for the user process. It holds all segments that have been received but not acknowledged for a particular connection. Movement of segments from this queue to the user receive buffer causes an ACK to be generated for the segments and the receive left window edge to be updated (see 8.3.9.). Holes (i.e. out-of-order segments) can be determined by comparing the left window edge with the sequence number of the first segment on the queue.

8.2.5.11.2. Queue and element structure

The major consideration here is to facilitate the detection of holes in a letter (out-of-order arrivals), so that everything arriving after the holes can be held. Segments following a hole should not be transferred to the to-user buffer queue or acknowledged. A complicating factor is that some controls use sequence number space, and thus cause a gap in the sequence number space. Elements with a special flag must be inserted in the queue to indicate that the hole is a result of a control arrival.

A reassembly queue element should include the segment sequence number, segment length, and a flag for control (padding) elements. The elements are ordered by sequence number. Using the current receive left window edge, the reassembler can detect holes as it removes elements from the head of the queue. Insertion, which will normally occur at the "tail", must always keep the queue ordered, so it may occur in the middle.

8.2.5.11.3. Access

ADD:

from-net segment handler (interprets and moves to reassembly queue, if necessary)

REMOVE:

reassembler (copy to to-user buffer)
cleanup and space manager

8.2.5.12. To-user buffer queue

8.2.5.12.1. The to-user buffer queue is "shared" by the TCP and the user process. The queue holds buffers that belong to the user, which were added to the queue by the user event receiver when a RECEIVE was performed. Buffers on this queue are unavailable to the user process until it is signalled that a letter or partial letter has arrived. The reassembler can use this buffer to hold any part of a letter, as long as the segments comprising the partial letter have been acknowledged. Under certain circumstances partial letters may be handed to the user. This could occur if some of the buffers for segments of the letter are preempted.

8.2.5.12.2. Queue and element structure

There is one instance of this queue per connection.

8.2.5.12.3. Access

ADD:

reassembler (copy into existing buffer)

REMOVE:

reassembler or to-user event sender (letter assembly complete)
cleanup

8.3. Functional Modules

8.3.1. The functional modules of the TCP are a composite set of TCP processes and subroutines. In conjunction with the data structures described previously, the module descriptions below help to present a comprehensive implementation model. In addition to discussing the function of the modules, each module description includes the modules to/from which calls are made and the data structures that the module accesses. Figure 6 is a useful diagram for viewing these interrelationships.

8.3.2. External Event Handler

8.3.2.1. The external event handler is the top-level process and thus the first module to run when the TCP is scheduled. It examines the TCP external event list and moves the events to an ordered list for access by the user and network event receivers. It then calls the user and network event receivers, using some fairness mechanism, so that events can be transformed into queue entries, intra-TCP signals, and/or subroutine calls.

8.3.2.2. After the event receivers run, the TCP scheduler is called.

8.3.2.3. The external event handler is the only module that is guaranteed to run when the TCP is scheduled. If the events are of the control nature and require excessive run time, the TCP scheduler may never be called.

8.3.2.4. Calls or signals

user event receiver, network event receiver, scheduler

8.3.2.5. Called or signalled by

none

8.3.2.6. Data structures accessed

external event list, ordered event list

8.3.3. User Event Receiver

8.3.3.1. The user event receiver is the user's (process) system call interface to the TCP and lower-level network modules. The user event receiver interprets external events from the user (process) and makes the appropriate call or signal on TCP functional modules.

8.3.3.2. The list of possible events include OPEN, CLOSE, SEND, RECEIVE, RETRACT, INTERRUPT, STATUS, STOP/RESUME TRANSMISSION, and GENERAL and are described in detail in Appendix B. All events will result in an immediate event back to the user (process) signifying that the event data (syntax) is acceptable or unacceptable. Generation of these events is the responsibility of the user event receiver; the events are sent by calls on the user event sender.

8.3.3.3. Later, various asynchronous events that correspond to the completion of some meaningful part of the activity may occur, but will be sent by other modules through calls on the user event sender.

8.3.3.4. Calls or signals

segmentizer, reassembler, control handler, space manager, user event sender

8.3.3.5. Called by

external event handler

8.3.3.6. Data structures accessed

ordered event list, intra-TCP signal board, TCB, from-user buffer queue, to-user buffer queue

8.3.4. Network Event Receiver

8.3.4.1. The network event receiver handles all events from the SIP, i.e. DATA, ERROR, and various control events. The network event receiver removes events from the ordered event list in order of precedence.

8.3.4.2. When the network event receiver gets a DATA event it adds the segment to the from-net segment queue. The queue element holds the address of the S-segment. The S-segment address must be returned to the SIP when the segment has been reassembled into a user letter buffer and ACKs sent to the foreign TCP. The handle on the segment is the S-segment address. Since the Binary Segment Leader and the T-segment header are fixed in size, the T-segment address or even the text address can be used to generate the address of the original S-segment.

8.3.4.3. Errors and control events from the SIP fall into two basic categories, TCP-SIP communication, and violations and problems.

8.3.4.4. TCP-SIP communication controls affect only the message passing mechanism and do not traverse the TCP, nor do they affect the state of any connection. These control events are handled by the network event receiver, which generates responding controls to the SIP by calling the network event sender.

8.3.4.5. S/P/T violations and network problems result in a call on the control handler.

8.3.4.6. Calls or signals

from-net segment handler, control handler, network event sender

8.3.4.7. Called or signalled by

external event handler

8.3.4.8. Data structures accessed

ordered event list, from-net segment queue

8.3.5. Scheduler

8.3.5.1. The scheduler allots the TCP's CPU cycles to the four processes--segmentizer, to-net segment handler, from-net segment handler, and reassembler.

8.3.5.2. The scheduling of TCP processes can affect throughput, delay, and resource availability. To control these, the scheduler must be able to affect the behavior of the processes it schedules. Since the scheduler does not know how to evaluate what a process has to do, it should be able to pass the maximum runtime available to the processes it schedules.

8.3.5.3. Calls or signals

segmentizer, to-net segment handler, from-net segment handler, reassembler

8.3.5.4. Called by

external event handler

8.3.5.5. Data structures accessed

Intra-TCP signal board

8.3.6. Segmentizer

8.3.6.1. The segmentizer is signalled by the user event receiver whenever a user (process) has performed a SEND. When the segmentizer is scheduled by the scheduler, it examines the from-user buffer queue of a specific TCB for old and new segmentizing work. For old work, it uses a pointer in the TCB to find where previous segmentizing was stopped.

8.3.6.2. The segmentizer must perform resegmentization when a letter has been preempted. In this case the from-user buffer queue element contains the original sequence number and text length so that the octets can be assigned their original sequence number.

8.3.6.3. Segmentizing is performed as the time allotted to the segmentizer permits. If a letter cannot be fully segmentized, progress is marked by setting a pointer in the TCB. When segmentizing is complete, the from-user buffer is moved to the segmentized buffer queue, where it remains until all segments of the letter are acknowledged.

8.3.6.4. Calls or signals

to-net segment handler, space manager

8.3.6.5. Called or signalled by

scheduler, user event receiver

8.3.6.6. Data structures accessed

from-user buffer queue, segmentized buffer queue, to-net segment queue, TCB

8.3.7. To-net Segment Handler

8.3.7.1. The to-net segment handler sends segments to the SIP, by calling the network event sender. The segments are taken from the to-net segment queue and/or the retransmit queue. Segments on the to-net segment queue have not been sent before; after their first transmission attempt, they are moved to the retransmit queue. Segments on the retransmit queue have been sent at least once and remain there until they have been acknowledged by the remote TCP.

8.3.7.2. The algorithm for choosing the next segment to send considers how the handler is signalled. The handler is signalled when a new to-net segment queue entry arrives, a retransmission timer times out, or an ACK or control needs to be sent.

8.3.7.3. All data segments that are transmitted have the latest receive left window edge in the ACK field. Error and control messages may have the ACK field set to refer to a received segment's sequence number, or no ACK (ACK bit off).

8.3.7.4. The to-net segment handler will check the to-net control queue and evaluate the state of the connection in the TCB to determine what, if any, errors or controls need to be transmitted. It must merge error and control segments with data segments from the to-net segment queue and the retransmit queue. Often this will require sending controls first, but will also involve some piggybacking of controls and data. Refer to section 9.6. for information concerning the assignment of sequence numbers to controls and the order of processing received controls.

8.3.7.5. The exact implementation of timers will be dependent on the hardware and the operating system. One could imagine several ways to determine which segment has timed out. One way is to set a hardware timer for the segment with the shortest timeout. Another would be to set a virtual timer that would accept some timer handle (sequence number) along with the timeout period and would pass the handle back when it timed out.

8.3.7.6. Retransmission policy is discussed in detail in section 9.3.

8.3.7.7. Once a segment has been selected for retransmission, its address is passed to the SIP and its timeout data is updated. No reordering of the queue is done. Unless the to-net segment handler is given a timeout handle (the sequence number) to the segment on the queue, it will have to search its queue for the segment with the next shortest timeout.

8.3.7.8. The total number of retransmissions must be monitored by the to-net segment handler, and if some maximum is exceeded, the user should be notified that the remote TCP is not responding and the connection should be aborted.

8.3.7.9. The to-net segment handler has special responsibilities when the "CLOSE in progress" flag in the TCB is set. In this case it will continue sending segments until it finds a segment with the EOL flag set, an empty to-net segment queue, and an empty from-user buffer queue. Then a FIN is sent to start the inter-TCP closing sequence.

8.3.7.10. Calls or signals

user event sender, network event sender

8.3.7.11. Called or signalled by

scheduler, segmentizer, control handler

8.3.7.12. Data structures accessed

to-net segment queue, to-net control queue, retransmit queue, TCB

8.3.8. From-net Segment Handler

8.3.8.1. The from-net segment handler is responsible for processing TCP segments as they arrive from the SIP. The segments are removed from the from-net segment queue and checked for the following: (1) valid TCB, (2) controls, (3) valid connection status, (4) ACKs, and (5) acceptable sequence numbers.

8.3.8.2. Errors and control, whether generated in the local or remote TCP, are passed directly to the control handler by call. Since some received controls occupy sequence number space, control segments should be copied to the reassembly queue for padding purposes.

8.3.8.3. For data and certain control segments, the segment's sequence number is compared against the current acceptability filter (see below).

8.3.8.4. Acceptability

8.3.8.4.1. The segment sequence number, the current acceptability filter size, and the receive left window edge determine whether the segment lies within the filter or outside of it.

Let F = acceptability filter

S = size of sequence number space

L = left window edge

R = (L+F) mod S = right filter edge

x = sequence number to be tested

For any sequence number, x, if

(R-x) mod S <= F

then x is within the filter.

8.3.8.4.2. A segment should be rejected only if all of it lies outside the filter. This is easily tested by letting x be, first the segment sequence number, and then the sum of segment sequence number and segment text length, less one. If the segment lies entirely to the left of the filter, and there are no segments waiting to be sent, then the from-net segment handler should call the control handler indicating that an ACK should be sent, even if a segment has to be created. The ACK will specify the current left window edge. This assures acknowledgment of all duplicates.

8.3.8.4.3. Notice that L + F is exactly the maximum sequence number ever "advertised" through the flow control window, plus one. This allows for controls to be accepted even though permission for them may never have been explicitly given. Of course, each time a control with a sequence number equal to the right filter edge is sent, the right filter edge must be incremented by one.

8.3.8.5. Any ACKs contained in incoming segments are passed by call to the ack'er, where the receive left window edge is updated and the segments are removed from the retransmit queue.

8.3.8.6. Successfully received data segments are placed on the reassembly queue in the appropriate sequence order, and the reassembler is signalled.

8.3.8.7. When the "CLOSE in progress" flag is set in the TCB the from-net segment handler will discard all data associated with incoming segments and will not acknowledge them. A special case is the arrival of a DSN (resynchronization request), which must be acknowledged even though it does not mean that all previous segments have been handed to the user.

8.3.8.8. Calls or signals

reassembler, control handler, ack'er

8.3.8.9. Called or signalled by

scheduler, network event receiver

8.3.8.10. Data structures accessed

from-net segment queue, reassembly queue, TCB

8.3.9. Reassembler

8.3.9.1. When the reassembler is scheduled it examines the reassembly queue for each TCB. If it is nonempty then it sees whether the to-user buffer queue is empty. If it is the reassembler, by a call to the user event sender, sends a

GENERAL message to the user soliciting a RECEIVE and goes on to the next TCB; otherwise if the first segment matches the left window edge, the segment can be moved into the to-user buffer queue. The reassembler keeps transferring segments into the user buffer until the letter is completely transferred, or something causes it to stop. Note that a buffer may be partly filled and then a sequence 'hole' is encountered in the receive segment queue. The reassembler must mark progress so that the buffer can be filled up starting at the right place when the 'hole' is filled. Similarly a segment might be only partially emptied when a buffer is filled, so progress in the segment must be marked.

8.3.9.2. If a letter was successfully transferred to a to-user buffer then the reassembler signals the user, through a call to the user event sender, that a letter has arrived. It then dequeues the buffer associated with it from the to-user buffer queue. If the buffer is filled then the user is signaled and the buffer dequeued as before. The event code indicates whether the buffer contains all or part of a letter.

8.3.9.3. In every case when a segment is delivered to a buffer, the receive left window edge is updated. This updating must take account of the extra octet included in the sequencing for certain control functions (SYN, INT, FIN, DSN). As the left window edge is updated, the segment is moved off the reassembler queue and "returned" to the SIP through a call to the network event sender.

8.3.9.4. Note that the reassembler should not work on a TCB for more than one to-user buffer's worth of time, in order to give all TCB's equal service.

8.3.9.5. When the "CLOSE in progress" flag in the TCB is set the reassembler will not attempt to reassemble any new segments and will return the receive buffers to the user (through a call to the user event sender) with a byte count indicating what has been reassembled at the time the CLOSE was received.

8.3.9.6. Calls or signals

control handler, user event sender, network event sender

8.3.9.7. Called or signalled by

user event receiver, scheduler, from-net segment handler

8.3.9.8. Data structures accessed

reassembly queue, to-user buffer queue, TCB

8.3.10. Control Handler

8.3.10.1. There are many sources of control messages that must be handled by the TCP. Controls can be categorized by where they originate, whether or not they change the state of the connection, and how far they travel with respect to the TCP. Some originate in the local TCP and result in messages to users or remote TCP's. Some cause state changes in the TCP, but never reach the next layer of protocol. Some controls traverse the TCP from boundary to boundary, with the TCP playing intermediary. The following control messages must be handled by the TCP:

8.3.10.1.1. SIP-to-user messages

network entity going down (handled by network event receiver)

segment rejection or transmission problem

8.3.10.1.2. Controls for TCP-SIP communication (handled by network event receiver)

8.3.10.1.3. TCP-to-TCP controls

connection specific controls

all connections for this TCP

8.3.10.1.4. User-to-TCP controls

single connection controls

host-wide, TAC-wide controls

8.3.10.1.5. User-to-SIP controls (handled by user event receiver)

host going down

stop/resume transmission

8.3.10.2. The control handler is the hub of control activity; receipt or generation of a control by any functional module results in a call on the control handler.

8.3.10.3. It is the responsibility of the control handler to manage the state of the connection; moreover it is the only module that changes the state. Other functional modules examine the state to determine their correct operation, but they never modify the state. The controls that cause state changes and the transitions between states are discussed at length in Appendix F.

8.3.10.4. Error controls are covered in the appendix for state transitions. They are also discussed at length in a separate section on Error Handling 9.7.

8.3.10.5. The control handler informs the to-net segment handler whenever it is appropriate to send a control. It does this by adding the control to the to-net control queue and signalling the to-net segment handler. The control handler does not decide where in the output stream the control will be placed. The to-net segment handler is responsible for this, which requires knowledge of controls in the to-net segment handler. The to-net segment handler decides which control is to be selected from the queue and transmitted next.

8.3.10.6. Calls or signals

user event sender, network event sender, to-net segment handler, ack'er, space manager, cleanup

8.3.10.7. Called by

user event receiver, network event receiver, from-net segment handler, reassembler

8.3.10.8. Data structures accessed

to-net control queue, TCB

8.3.11. Ack'er (Repackager and Timer Resetter)

8.3.11.1. The ack'er has several responsibilities which should be assumed in the following order:

8.3.11.1.1. Remove acknowledged segments.

The ack'er is called by the from-net segment handler whenever an ack is received that exceeds the current send left window edge. The ack'er must remove all acknowledged sequence numbers from the retransmit queue, including any timer bookkeeping that may exist outside the queue.

8.3.11.1.2. Repackage partially acknowledged segments or shrunken window victims.

The receiving TCP may acknowledge any part of a segment, though it will acknowledge full segments whenever possible. The TCP should be able to repackage a partially acknowledged segment, especially if less than half of the original segment was acknowledged.

When receive buffer space is low in the receiving TCP, partial acknowledgement may be coupled with a reduction in the window size to attempt to slow the flow. Repackaging should be done so that all new segments are no larger than the new window size.

Repackaging is accomplished by removing the segment from the retransmission queue and reinserting it after it has been broken into smaller segments. The range of sequence numbers does not change for a repackaged segment.

8.3.11.1.3. Remove segments from the segmentized buffer queue when the EOL segment has been acknowledged and send the user (through the user event sender) an event to acknowledge the letter.

As each segment on the retransmit queue is acknowledged and removed, it is checked for the EOL (End-Of-Letter) bit. When an EOL is received, in order, all segments of a letter have been acknowledged and the ack'er removes the from-user buffer from the segmentized buffer queue. The buffer is "returned" to the user by sending an event indicating that the letter has been received by the destination TCP.

8.3.11.1.4. Reset timeout data on items in the retransmit queue.

Certain SIP controls indicate that an error has occurred in the network that has prevented the successful transmission of a segment, e.g. incomplete transmission. When one of these occur, the ack'er should reset the segment's timeout so that it will be retransmitted at a rate consistent with the error encountered.

Resetting retransmission timeouts may also be required if flow control is exerted by the destination, i.e. a shrinking window is encountered.

8.3.11.2. Calls or signals

space manager, user event sender

8.3.11.3. Called or signalled by

from-net segment handler, control handler

8.3.11.4. Data structures accessed

retransmit queue, segmentized buffer queue, TCB

8.3.12. Space Manager

8.3.12.1. The TCP space manager is responsible for the allocation and deallocation of all segment buffers for all the TCP functional modules. As described earlier, the space manager acquires its space from a global resource manager, e.g. an operating system resource allocator. During normal operation buffers are consumed by the segmentizer and released by the ack'er. They may also be consumed by the event senders and to-net segment handler when they generate control events and control segments. During connection closing, flushing of buffers may occur as cleanup is called.

8.3.12.2. Preemption of previous allocations must occur if the space manager's available buffers are depleted and Category I activity requires buffer space. This will be accomplished by checking low precedence connections first, and for a chosen connection, segments on the to-net segment queue (i.e. segments that have not been sent yet) will be the first candidates for preemption (see 9.5.4.). Only if all buffer resources are allocated to Category I connections will the space manager refuse a Category I request.

8.3.12.3. If preemption of a segment occurs, the entire letter from which it came will be resegmentized. If controls are involved, they will be entered on the to-net control queue if they have not been sent and on the retransmit queue if they have been sent. The controls appear as new, separate segments with their original sequence numbers.

8.3.12.4. It is assumed that the TCP space manager will be called to free all resources that it has allocated. That is, no resources will be lost to the TCP pool by passing buffers to its neighbors (user or SIP). An overall strategy must be used that assures that buffer resources do not "leave" a protocol module. Thus it is expected that the protocol modules will acquire space from a global resource manager and this manager will insure that no protocol module will acquire more than its fair share.

8.3.12.5. Calls or signals

preempter, user event sender, control handler

8.3.12.6. Called by

segmentizer, user event sender, network event sender, to-net segment handler, control handler, ack'er

8.3.12.7. Data structures accessed

All queues, TCB

8.3.13. Cleanup

8.3.13.1. Cleanup is called when a connection must be closed. The closing may be the result of a user request, remote user or TCP request, network problems, or resource preemption. Closing can occur in either a flushing (Immediate) or nonflushing (Deferred) mode.

8.3.13.2. An Immediate CLOSE requires immediate release of all resources, except the TCB, associated with a connection. Given a TCB, cleanup frees all buffers associated with a connection. It returns all buffers to the user (process) or the SIP, with the appropriate event data to indicate the reason. When the appropriate TCP control messages have been exchanged, the TCB is deleted as well.

8.3.13.3. Calls or signals

user event sender

8.3.13.4. Called or signalled by

control handler

8.3.13.5. Data structures accessed

all queues, TCB

8.3.14. User Event Sender

8.3.14.1. The user event sender is responsible for all events to the user generated by the local TCP. It calls upon the host, TAC, or CCU event sending mechanism to post events for the user (process). Events are generated in most of the TCP modules in an out-of-line fashion, but of course, cannot take effect until the current TCP process completes execution.

8.3.14.2. Called or signalled by

user event receiver, to-net segment handler, reassembler, control handler, ack'er, preempter, cleanup

8.3.15. Network Event Sender

8.3.15.1. The network event sender is called by the functional modules when events destined for the SIP must be sent. It then calls upon the host, TAC, or CCU event sending mechanism to post events for the user (process) or SIP. Events are generated in most of the TCP modules in an out-of-line fashion, and of course, cannot take effect until the current TCP process completes execution.

8.3.15.2. Called or signalled by

net event handler, to-net segment handler, reassembler, control handler

8.4. States and Transitions

8.4.1. To aid in the description of the TCP implementation model, a state diagram, Figure 7, and a listing of state transitions is included in Appendix F.

TCP CONNECTION STATES

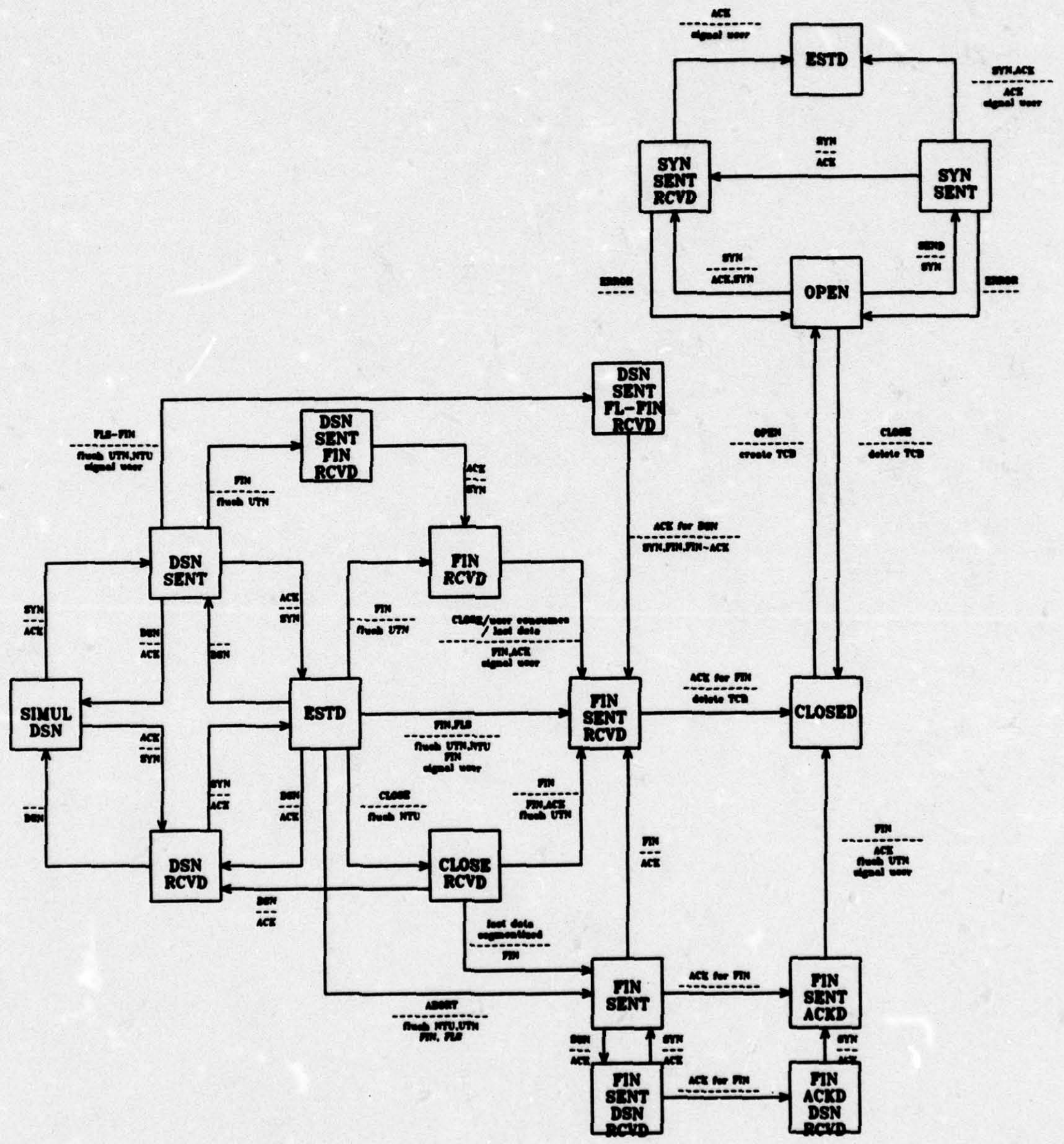


Figure 7

9. TCP RELIABILITY AND EFFICIENCY

9.1. Connection Management

9.1.1. Opening

9.1.1.1. When a user requests that a connection be opened, only the bookkeeping data block (TCB) for the connection is created. No network traffic is generated in response to the OPEN request. Connection synchronization, and thus true connection establishment, begins when a waiting TCB receives its first (synchronizing) message.

9.1.1.2. OPENs are distinguished by the completeness with which the address of the remote correspondent, the send precedence, and the TCC for the connection are specified. Notice that security cannot be unspecified in an OPEN.

9.1.1.3. TCP supports opening of connections in which some part of the destination address, security, send precedence, minimum receive precedence, or TCC is left unspecified. Such connections are called LISTENs. The unspecified entities for a LISTEN connection become bound when a message for that connection arrives. The bound entities are passed to the LISTENing process, at which time it can close the connection if the entities are unsatisfactory.

9.1.1.4. By the nature of a LISTENing process (usually some kind of logger or server), it is assumed that the process is willing to wait indefinitely for a remote connection request. Thus the process should not expect the TCP to communicate with it unless a message for that connection arrives. If a watchdog timer is appropriate for the connection, it should be provided by the user.

9.1.1.5. To allow the user to LISTEN for any one of several destinations by address or to LISTEN for any one of several user groups (TCC), the user may specify a list of destination addresses or a list of TCC's in the open request.

9.1.1.6. A user may not send on a LISTENing connection that has not been established, since no incoming message has bound its unspecified parameters.

9.1.1.7. An open in which the foreign user address, TCC, and send precedence are fully specified is called a CONNECT. A CONNECT is used when strict selectivity of the correspondent is desired. The user is not notified explicitly when such a connection becomes established.

9.1.2. Synchronization and Resynchronization

9.1.2.1. Initial sequence number selection

9.1.2.1.1. The protocol places no restriction on a particular connection being used over and over again. New instances of a connection will be referred to as incarnations of the connection. The problem that arises owing to this is, "how does the TCP identify duplicate segments from previous incarnations of the connection?". This problem becomes apparent if the connection is being opened and closed in quick succession, or if the connection breaks with loss of memory and is then reestablished.

9.1.2.1.2. The essence of the solution [reference 2] is that the initial sequence number (ISN) must be chosen so that a particular sequence number can never refer to an "old" octet. Once the connection is established the sequencing mechanism provided by the TCP filters out duplicates.

9.1.2.1.3. For the connection to be established or initialized, the two *TCP's must synchronize on each others* initial sequence numbers. Hence the solution requires a suitable mechanism for picking an ISN, and a slightly involved handshake to exchange the ISN's. A "three way handshake" is necessary because sequence numbers are not tied to a global clock in the network, and TCP's may have different mechanisms for picking the ISN's. The receiver of the first SYN has no way of knowing whether the segment was an old delayed one or not. Unless it remembers the last sequence number used on the connection, which is not always possible, it must ask the sender to verify this SYN.

9.1.2.1.4. The "three way handshake" and the advantages of a "clock-driven" scheme are discussed in [reference 2]. More on the subject, and algorithms for implementing the clock-driven scheme can be found in [reference 3].

9.1.2.2. Establishing (synchronizing) a connection

9.1.2.2.1. The "three way handshake" is essentially a unidirectional attempt to establish the connection, i.e. there is an initiator and a responder. The TCP's should, however, be able to establish the connection even if a simultaneous attempt is made by both TCP's to establish the connection.

9.1.2.2.2. The example below indicates what a three way handshake between TCP's A and B looks like:

```
A                                              B

-->            <SEQ x><SYN>                  -->

<--         <SEQ y><SYN, ACK x+1>            <--

-->    <SEQ x+1><ACK y+1><DATA BYTES>        -->
```

9.1.2.2.3. The receiver of a "SYN" is able to determine whether the "SYN" was real (and not an old duplicate) when a positive "ACK" is returned for the receiver's "SYN,ACK" in response to the "SYN". The sender of a "SYN" gets verification on receipt of a "SYN,ACK" whose "ACK" part references the sequence number proposed in the original "SYN". A more detailed discussion about unacceptable SYN's and other errors encountered while synchronizing a connection is given in section 9.7.

9.1.2.2.4. When establishing a connection, the state of the TCP is represented by four states (see Figure 7) --

OPEN - synchronizing did not start.

SYNsent - local TCP initiated synchronization

SYNsent-rcvd - sequence numbers established.

ESTD - connection established.

9.1.2.2.5. Notes:

A user may be in the OPEN state if the synchronization of the connection was never attempted or if a previous synchronization attempt failed.

In the SYNsent-rcvd state the received SYN has been acked. TCP will never, while establishing a connection, send a SYN without an ACK in response to a received SYN. However, the local TCP does not assume identical behavior from the remote TCP and is ready to accept separate SYN and ACK.

The SYNsent-rcvd state is reached when both TCP's have made a simultaneous attempt to synchronize the connection.

The receipt of unacceptable SYN's while in the synchronizing procedure can result in a reset of the connection to the OPEN state; thus the parameters provided by the user must be retained until the ESTD state is reached. A reset cannot occur once the ESTD state has been reached, thus the bound parameters only need be kept in the TCB. For more discussion of error handling while synchronizing see 9.7.

9.1.2.2.6. The example below demonstrates an attempt to establish a connection. The state of the connection is indicated when a change occurs. We specifically do not show the cases in which connection synchronization is carried out with packets containing both SYN and data. We do this to simplify the explanation, but we do not rule out an implementation which is capable of dealing with data arriving in the first packet (it has to be stored temporarily without acknowledgement or delivery to the user until the arriving SYN has been verified).

9.1.2.2.7. The "three way handshake" now looks like:

```
      A                                          B

OPEN                                          OPEN

        -->        <SEQ x><SYN>        -->

SYNsent                                       OPEN

        <--   <SEQ y><SYN, ACK x+1>   <--

SYNsent                                SYNsent-rcvd

        --> <SEQ x+1><ACK y+1><DATA> -->

ESTD                                          ESTD
```

9.1.2.3. Half-open connections [reference 4]

9.1.2.3.1. An established connection is said to be a "half-open" connection if one of the TCP's has closed the connection at its end without the knowledge of the other, or if the two ends of the connection have become desynchronized due to a crash that resulted in loss of memory. Such connections will automatically become reset if an attempt is made to send data in either direction. However, half-open connections are expected to be unusual, and the recovery procedure is somewhat involved.

9.1.2.3.2. If one end of the connection no longer exists, then an attempt by the other user to send any data on it will result in the sender receiving the event code "Connection does not exist at foreign TCP". Such an error message should indicate to the user process that something is wrong and it is expected to CLOSE the connection.

9.1.2.3.3. Assume that two user processes A and B are communicating with one another when a crash occurs causing loss of memory to B's TCP. Depending on the operating system supporting B's TCP, it is likely that some error recovery mechanism exists. When the TCP is up again process B is likely to start again from the beginning or from a recovery point. As a result B will probably try to OPEN the connection again or try to SEND on the connection it believes open. In the latter case it receives the error message "connection not open" from the local TCP. In an attempt to establish the connection B's TCP will send a segment containing SYN. A's TCP thinks that the connection is already established and so will respond with the error "unacceptable SYN (or SYN/ACK) arrived at foreign TCP". B's TCP knows that this refers to the SYN it just sent out, and so it should RESET the connection and inform the user process of this fact.

9.1.2.3.4. It may happen that B is passive and only wants to receive data. In this case A's data will not reach B because the TCP at B thinks the connection is not established. As a result A's TCP will timeout and send a QRY to B's TCP. B's TCP will send STATUS saying the connection is not synched. A's TCP will treat this as if an implicit CLOSE had occurred and tell the user process, A, that the connection is closing. A is expected to respond with a CLOSE command to his TCP. However, A's TCP does not send a FIN to B's TCP, since it would not be accepted anyway on the unsynchronized connection. Eventually A will try to reopen the connection or B will give up and CLOSE. If B CLOSEs, B's TCP will simply delete the connection since it was not established as far as B's TCP is concerned. No messages will be sent to A's TCP as a result.

9.1.2.3.5. Half-open connections are expected to be quite rare. One possible source, other than a system crash, might be when a resynchronization fails because the SYN holding A's new sequence number does not get acknowledged properly. This suggests that an old SYN was acknowledged by B's TCP or an old ACK arrived at A's TCP, making it impossible for A's TCP to know if B's TCP knows A's sequence numbers.

9.1.2.4. Resynchronization [reference 5]

9.1.2.4.1. The basic sequence numbering strategy employed in the TCP is to assign (implicitly) a sequence number of each octet (8 bit unit) of text transmitted in a segment. [reference 4,6] As explained in Tomlinson and amplified by Dalal [reference 2,3], the finiteness of the available sequence number space requires that after a time sequence numbers must be re-used.

9.1.2.4.2. Care must be taken that the available sequence number space not be used up so fast that some segments carrying old sequence numbers are still in the network when these numbers are re-used. To assure this, we assume that segments can only remain in the network for a certain maximum lifetime. The maximum rate at which sequence numbers are used is related to this lifetime and to the size of the sequence number space. If the lifetime is T seconds and the maximum bandwidth (rate of sequence number consumption) is B, octets/second and the size of the sequence space is S octets, then we must have (see [reference 6])

$$S > 2*B*T$$

9.1.2.4.3. Indeed, it may happen that T is really a random variable without a guaranteed maximum, in which case reliability can be improved by writing

$$S = K*B*T$$

which, for sufficiently large K makes the probability nearly zero that two segments carrying different text are assigned the same or overlapping sequence numbers.

9.1.2.4.4. The following parameters have been chosen for initial implementation:

S = 2**32 octets	= sequence number space
B = 2**18 octets/sec (approx. 2 MB/sec)	= bandwidth
T = 2**5 seconds (approx. 0.5 minutes)	= maximum segment lifetime (estimate)

9.1.2.4.5. The minimum time to cycle through the sequence number space (C) is

C = S/B = 2**14 sec (approx. 4.55 hours)

9.1.2.4.6. Thus a sequence number space of 2**32 octets assures us that unless a maximum segment lifetime is greater than 2.27 hours, we do not run the risk of assigning the same sequence number to two different octets. Of course, this requires that the full sequence number space has been cycled through before a duplicate number is assigned.

9.1.2.4.7. Earlier mention was made of the need to select an initial sequence number (ISN) using some well-known, periodic value, such as the time of day. Figure 8 plots the ISN value against time, where time and ISN are measured in the same units (e.g. using bandwidth (B) as a parameter).

9.1.2.4.8. The same sequence number cannot be used to start each connection, since this might introduce unintentional duplicate sequence numbers. We cannot rely on remembering the last sequence number used on a given connection, since this information may have been lost (e.g. host failure). Even during the course of a particular connection, it may become necessary to resynchronize sequence numbers to avoid potential failures.

9.1.2.4.9. In Figure 9, we show the history of sequence numbers used by a particular connection. The lines labeled "ISN" represent the maximum permitted rate at which sequence numbers can be used, however, this may be different than the maximum throughput rate for the TCP.

THE NEED FOR RESYNCHRONIZATION

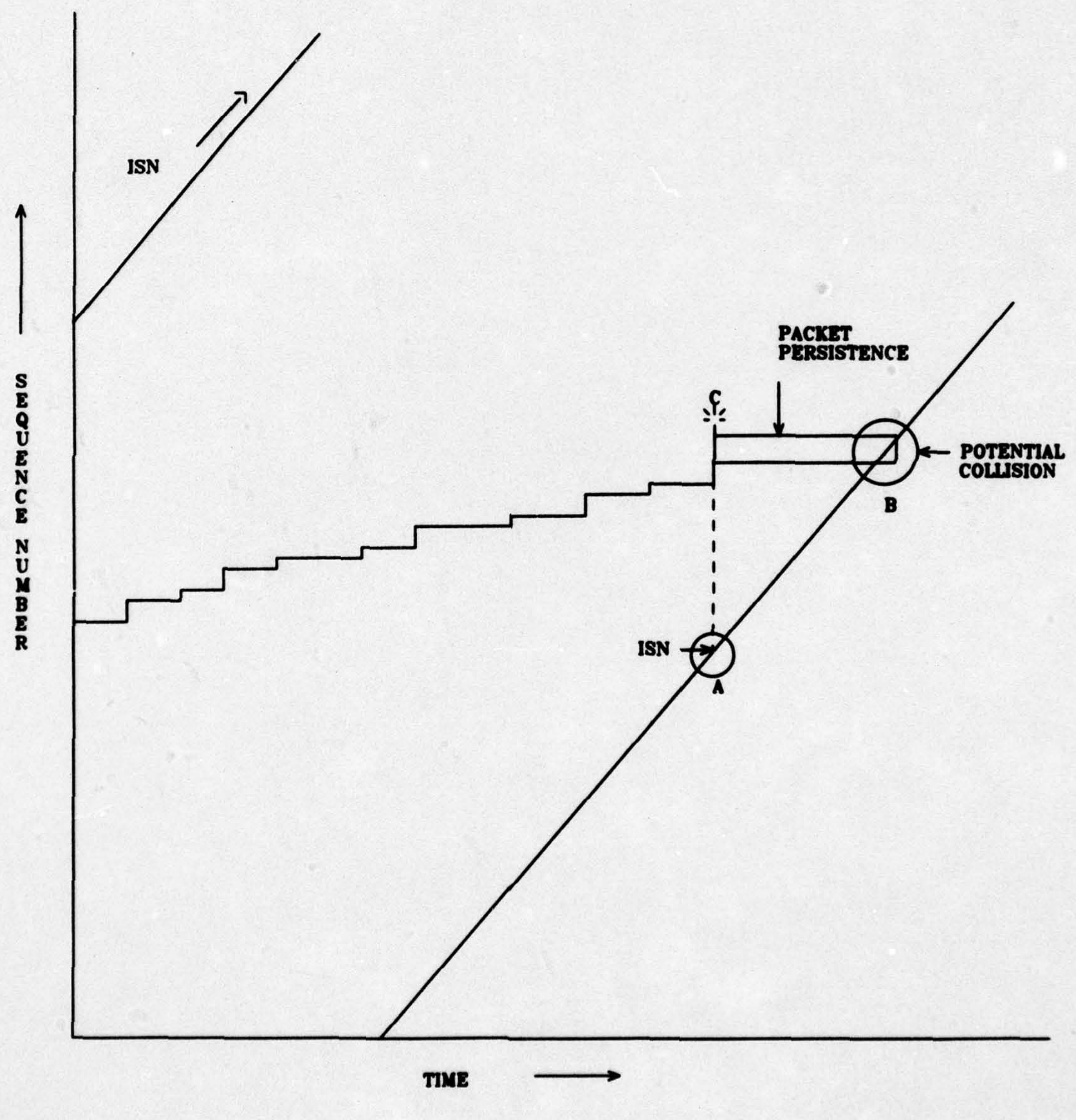
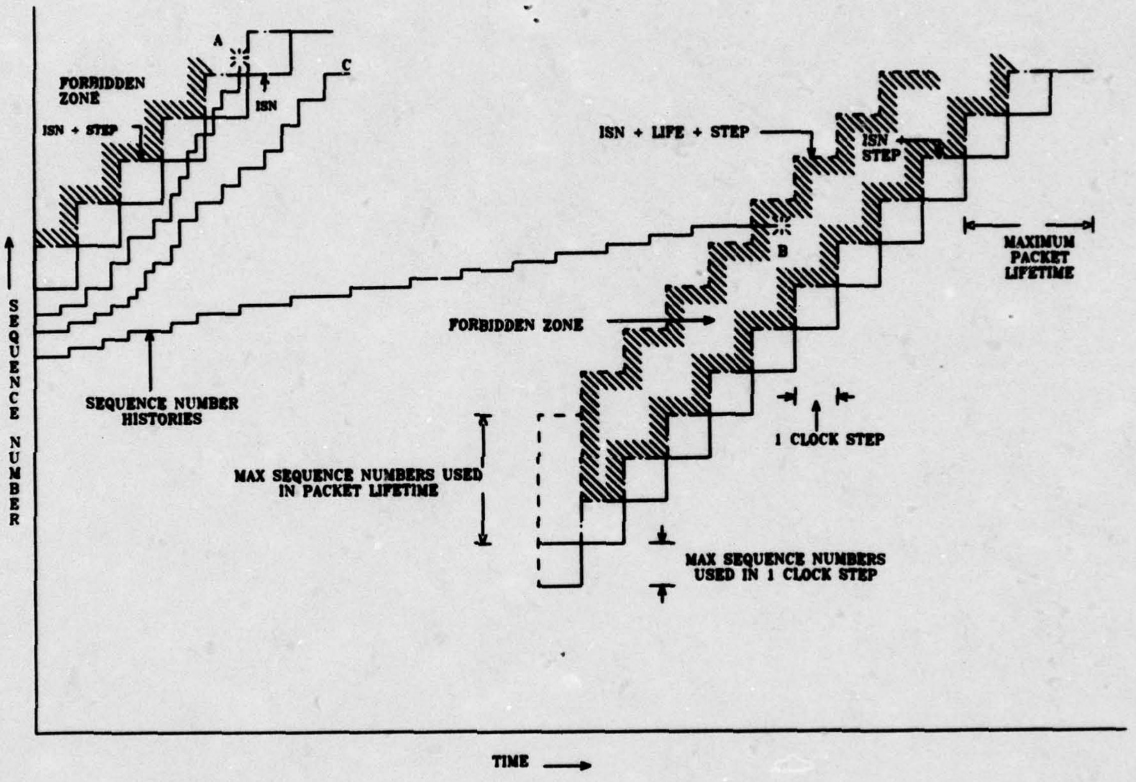


Figure 8

DETECTING THE FORBIDDEN ZONE



LEGEND

_⌐	ISN
- - -	ISN + LIFETIME + STEP
-·-	ISN + STEP

Figure 9

9.1.2.4.10. Suppose that the TCP supporting the connection fails at "C" and must be restarted. Assume, also, that the sequence number selected to restart is drawn from the value of ISN at the time event "C" occurred. The shaded area between "C" and "B" represents the maximum expected time that segments emitted at "C" can stay in the net. Clearly, the ISN line intersects this shaded area, indicating that, after the restart, it is possible that segments emitted at "C" may become undistinguishable from those potentially emitted along the ISN curve. To correct this flaw, the sequence number currently to be used on the connection must be resynchronized before running into the forbidden zone to the left of the ISN line.

9.1.2.4.11. Testing for the need to resynchronize

As segments are produced and sequence numbers assigned to them, the TCP must check for two possible conditions which indicate that resynchronization is needed. The first is that sequence numbers are being used up so fast that they advance faster than ISN. The other is that they advance so slowly that ISN "catches up with them."

The basic method of selecting an initial sequence number is to delay for an *arbitrary period* labelled a "clock tick" or STEP and then select the new ISN. In the TCP implementation, this amounts to one second of time during which up to 2**18 octets may be transmitted. So,

STEP = 1 second (2**18 octets)

In Figure 9, three sequence number histories are traced, ending in points "A", "B", and "C". In the trace labelled "A," sequence numbers are used at such a rate that point "A" lies beyond ISN plus one STEP. If the connection were to fail and be restarted at "A," the new ISN would be just below point "A" and would introduce potential unwanted duplicates.

This situation can be detected before transmission of the segment. Let L be the length of the data in octets. Let SEQ represent the proposed sequence number of the segment, and SEQ+L-1 be the sequence number implicitly associated with the last octet of segment data. If ISN+STEP (at the moment that SEQ is to be assigned) lies in the range [SEQ, SEQ+L-1], then the type "A" ISN failure is about to occur. The solution is to send only as much text as is allowed (which does not result in the failure) and WAIT for the clock to tick again.

The situation in curve "B" is quite different. In this case, the connection is using numbers so slowly that the forbidden zone preceding the ISN curve has advanced and run into the connection sequence number curve. There are two solutions. One is to wait for the segment lifetime plus one clock step to expire (in which case the sequence history will pop out of the forbidden zone again). The other is to actively resynchronize the connection. The test for the type "B" situation is whether sequence number SEQ lies in the range [ISN, ISN+T+STEP].

Note that all tests for inclusion must be modulo S to account for the wrap around of sequence numbers.

Curve "C" in Figure 9 shows a sequence number trace which tends, on the average, to lie within legal values at all times.

9.1.2.4.12. Resynchronizing controls

A special TCP *control*, the DSN, is sent by the local TCP to initiate resynchronization of the local send direction of the connection. It is assigned a sequence number from the old sequence number space and is handled in-stream. Acknowledgement of the DSN by the remote TCP is implicit acknowledgement that all segments preceding it were successfully delivered. Normally a DSN will be acknowledged, after which a SYN is sent by the local TCP with a newly chosen sequence number.

The DSN must be sent prior to entering the "forbidden zone," so that a new sequence number can be selected using the ISN curve without delay. A periodic check is made on each connection to insure that the "forbidden zone" is not entered without first resynchronizing during a period of inactivity. This check may result in the resynchronization of the connection, i.e. sending a DSN and selecting a new sequence number, even if no other segments are being sent.

A DSN should never be flushed. It must contain the last sequence number of the old set of sequence numbers. Even if acknowledgement of the DSN is delayed, as in the case of a zero receive window in the receiver (see Appendix H), flushes must be inhibited until the acknowledgement for the DSN is received.

9.1.2.4.13. The ISN lines

The ISN lines in Figure 8 indicate the maximum rate at which sequence numbers can be consumed for a particular connection. The slope of the ISN lines are related to the maximum bandwidth (B), the minimum time between resynchronization for a connection of low activity (RT), the size of the sequence number space (S), and the maximum network segment lifetime (T).

Since sequence numbers cannot be consumed faster than the transmission medium can pass data,

ISN slope (in octets/sec) =< B / 8

So that the forbidden zone will not overlap adjacent ISN lines we have

T << RT

Initially, we recommend the time to resynchronize, RT, to be 2**12 seconds or approximately one hour. The relationship between RT and the ISN slope is

RT = (S / (ISN slope)) - T

The ideal slope of the ISN lines would be the exact rate of sequence number consumption on the connection, so that forced resynchronization would never occur. This is clearly impossible, but for each implementation a different slope may be appropriate. The constraints listed above should be considered in selecting the slope.

9.1.2.4.14. Conclusions

The value of clock step may be arbitrarily small, depending on the delay overhead to wait for the tick.

Checking for resynchronization is simple:

if SEQ is in the range [ISN+T+STEP] then resynchronize.

if ISN+STEP is in the range [SEQ, SEQ+L-1] (where L = segment length in octets) then wait for the clock to tick.

9.1.3. S/P/T Management

9.1.3.1. The TCP assumes a heavy role in S/P/T management when it establishes connections with remote TCP's. The role is played in two distinct ways, the checking of initial connection values for S/P/T and the monitoring of the S/P/T of each message to insure that any changes that have occurred are legal. In the following it is assumed that the aforementioned authorization table of S/P/T information is accessible by the TCP.

9.1.3.2. Security

9.1.3.2.1. The security levels for the two directions of data flow of a connection can be different. Moreover, the user (process) may designate the security for a connection as an absolute (single level) or as a ceiling. A ceiling value defines the maximum security level that can be sent over a connection. The security levels of messages on a connection may vary, provided that they fall within the range of security levels allowed for that direction of that connection.

9.1.3.2.2. When a ceiling security level has been specified, controls that are not piggybacked on data will be transmitted at the lowest security level allowed by the authorization table.

9.1.3.2.3. For a LISTENing open, the security level may be unspecified by the user. The send security will be set to that of the received connection establishing segment, provided it is allowed by the authorization table. The TCP will check a specified security against the authorization table to prevent indefinite waiting for connections opened in LISTENing mode with illegal security.

9.1.3.3. Precedence

9.1.3.3.1. The precedence of a connection is managed so that the precedences in the two directions are equal, if not restricted by the authorization table or packet switch. The TCP's will both attempt to establish the connection such that the precedences in both directions are equal to the maximum of the precedences requested by the local and remote users. Since one user may specify a precedence higher than both the send precedence requested by the remote user and the maximum allowed for that user in his authorization table or packet switch, certain connections may have unequal precedences for the two directions of flow. The send precedence level is always restricted by the maximum level allowed by the authorization table or packet switch.

9.1.3.3.2. In the case when the connection establishing message is received and the precedence level of that message exceeds the maximum send precedence level of the receiver, the reply will be made at the level of the original message, if not restricted by the authorization table or packet switch. That is, the TCP will give the reply the highest precedence level it can, up to that of the original message, even though the receiver's initial connection request was at a lower level. The user will be notified with a special event if the send precedence level for the connection has been altered by the TCP.

9.1.3.3.3. For certain applications it is desirable to be able to LISTEN with a minimum receive precedence so that a remote attempt to establish a connection at a low precedence will be rejected. The minimum receive precedence parameter is optional and defaults to the minimum precedence allowable.

9.1.3.3.4. In the case when the send precedence is unspecified in a LISTENing open, the local user's precedence becomes bound by the first precedence received, within the constraints mentioned above. The local user is notified and passed the new precedence.

9.1.3.3.5. The TCP will never establish a connection, by sending the initial message, at a precedence level higher than that requested by the user.

9.1.3.3.6. The precedence levels for a connection will be monitored to ascertain that they remain unchanged for the lifetime of the connection.

9.1.3.4. TCC (user groups)

9.1.3.4.1. The main motivations for having TCC checking at the TCP level are (1) to allow TCC assignment at the individual user (process) level as well as the subscriber (host) level, (2) to permit host administrators to assign stricter TCC's to their users, and (3) to prevent indefinite waiting for LISTENing connections when the TCC is illegal.

9.1.3.4.2. The implementation model is characterized by having only one TCC per established full-duplex connection, i.e. the receive TCC equals the send TCC. Unlike the PS, the TCP applies TCC checking to a specific connection, providing connection establishment management and connection monitoring.

9.1.3.4.3. If the TCC is specified in an open, either as a list of TCC's or a single TCC, the first TCC received from the remote TCP must match one of those specified, else the local TCP will reject the connection attempt. The local user will not be notified that an illegal attempt to connect was made. The remote, offending user will be notified that the connection has failed due to an unacceptable TCC.

9.1.3.4.4. In the case where the TCC is unspecified in a LISTENing open, the local user's send TCC becomes bound by the first TCC received. The local user is notified and passed the new TCC.

9.1.3.4.5. There is one TCC field in the TCB. It is first used as a pointer to the TCC list provided by the user, and later, when the connection becomes bound, is used for monitoring the send and receive TCC. Once the connection has been established, the TCC must remain constant for the lifetime of the connection.

9.1.3.4.6. In the event that a bad TCP allows a violation of user group, the receiving TCP will respond with a message indicating a violation and malfunction in the sending TCP. The connection will be closed.

9.1.4. Closing

9.1.4.1. Connections may be closed by the TCP in response to a number of conditions, including request by the user (process), error conditions on the connection, connection closing by the remote TCP, request for "Stop Transmission", and the initiation or termination of Periods Processing modes.

9.1.4.2. Two types of closes seem to be necessary, one for the immediate flushing of a connection and one for supporting a deferred, more graceful close.

9.1.4.3. A flushing close causes immediate shutdown of the connection as soon as both TCP's exchange the proper control messages. No more letters are sent and the user is not given any more incoming letters or partial letters. User buffers previously passed to the TCP for sending and receiving are returned to the user.

9.1.4.4. A deferred close causes all outstanding letters to the network to be sent and acknowledged, and all letters and partial letters that have been received by the TCP prior to the close to be given to the user (assuming the user has provided a buffer for them). When this has been done, the TCP sends a control message to initiate connection closing.

9.1.4.5. More specifically, the deferred close is started by the control handler. It flushes the network-to-user data by returning to-user buffers, including any reassembled segments completed prior to the CLOSE request. It then sets a flag in the TCB that signifies a graceful close is in progress. This flag is checked by the functional modules and the following action is taken when each is activated:

9.1.4.5.1. The user event receiver will reject SEND attempts with a "connection closing message."

9.1.4.5.2. The to-net segment handler will continue sending segments until it finds a segment with the EOL flag set, an empty to-net segment queue, and an empty from-user buffer queue. Then a FIN is sent to start the inter-TCP closing sequence.

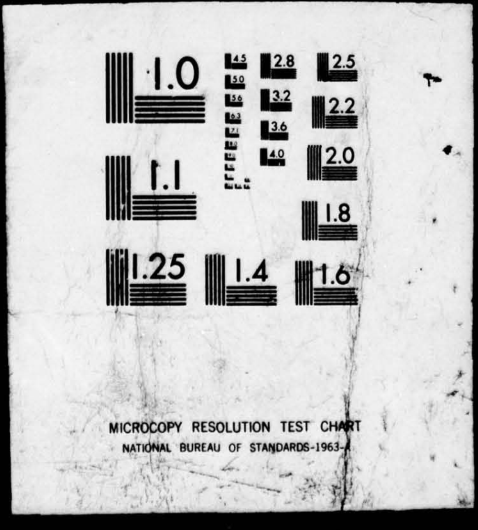
9.1.4.5.3. The from-net segment handler will discard all data associated with incoming segments and will not ACK them. A special case is the arrival of a DSN, which must be acknowledged.

9.1.4.5.4. The reassembler will not attempt to reassemble any new segments and will return the receive buffers to the user (through a call to the user event sender) with a byte count

IFIED

2 OF 2

AD A035337



1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3
7.1
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

indicating what has been reassembled at the time the CLOSE was received.

9.1.5. Multiple Connections

9.1.5.1. There is no restriction, beyond the availability of resources, on the number and nature of connections that a user (process) can have open at one time. Multiple connections may arise with either a common local socket or with different local sockets.

9.1.6. Transfer of Ownership

9.1.6.1. When a process decides to transfer ownership of a connection to another process within the same host it has to notify both the new owner and the TCP. The original owner must know the new owner's process ID and that the new owner intends to accept the connection.

9.1.6.2. When the original owner is ready to transfer the connection a MOVECONN event is sent to the TCP indicating the LCN of the connection and the process ID of the new owner. The TCP immediately checks the authority of the new owner to use the connection. If the authority is verified all buffers will be returned to the original owner and the new process ID will be recorded in the TCB. The connection is now owned by the new process. If the new owner is not authorized to use the connection the MOVECONN is rejected and ownership is not transferred.

9.1.6.3. There is a potential race condition here that is solved by first notifying (or creating) the new owner, sending the MOVECONN event, and finally signalling (or thawing) the new owner.

9.1.6.4. It should be noted that at no time is the connection not owned by a process. If the owner process fails in some way the connection will be closed in a graceful manner.

9.2. Scheduling and Preemption

9.2.1. As mentioned in Operating System Requirements, the TCP will be responsible for limiting its execution time, so that other modules in the TAC or CCU receive a fair portion of CPU cycles. Scheduling of processes within the TCP will be the responsibility of the TCP, specifically the scheduler module. Each process must perform only one task, representing the response to only one event, and then return to the scheduler. A process runs uninterrupted until it returns to the scheduler, where a check is made for new events for the processes and a new process is chosen to run.

9.2.2. Since a process runs uninterrupted, no preemption is done by the scheduler. High precedence events will be serviced as soon as the scheduler gains control, since it will always schedule a process with high precedence events on the ordered list first. It will be the responsibility of the process to examine all its pending work and respond to high precedence events first.

9.2.3. Each TCB resides on one of two chains, a high precedence chain or a low precedence one. The chains can be used to locate pending activity, since each TCB has pointers to pending work queues. There are two globals available to all TCP functional modules, a pointer into the high precedence chain and a pointer into the low precedence chain. (In the future it may be advantageous to have a pair of pointers for each of the four inferior processes so that fairness among connections can be efficiently administered.)

9.3. Retransmission Policy

9.3.1. Retransmission of segments is necessary to reliably deliver segments in an environment where segment discarding and positive acknowledgement are used. However, the frequency of retransmission can have a significant effect on the achievable throughput for a single connection and the achievable throughput for all neighboring (using the same channel) connections.

9.3.2. According to Sunshine [reference 4] the best strategy for choosing a retransmission timeout is to set the timeout equal to the time it would take to successfully transmit a segment and receive an ACK for it, plus a little, if there were no lost or damaged segments. This policy weighs the advantage of "keeping the pipe full" against the disadvantage of reducing the bandwidth with unnecessary retransmissions. It essentially means that a retransmission is undesirable unless there is a high probability that the segment was discarded.

9.3.3. Therefore, the retransmission timeout should be set once to

R = Mean TCP-to-TCP round trip time + epsilon

9.4. Flow Control

9.4.1. Flow control is a concern of the TCP at three interfaces, the TCP-user, the TCP-SIP, and the TCP-TCP. In all cases, high precedence (Category I) traffic will have priority and should never be preempted, denied resources, or slowed by flow control, unless all available resources are currently allocated to other Category I activity.

9.4.2. It is the responsibility of the user (process) to prevent the user terminal from swamping it with outgoing messages. The TCP will simply reject requests from the user (process) that it cannot service because of low resources.

9.4.3. The SIP will control the flow from the TCP to it, using rejection for non-Category I traffic. The TCP will not control the flow from the SIP, but it may discard traffic after checking it for control or high precedence level data. This asymmetry between protocol levels is possible because there is flow control and letter acknowledgement at the higher (TCP-TCP) level. A host does not need to say "slow down" to the packet switch, because it can say "slow down" to the remote source. If necessary, it can discard any segment it receives from the switch, since it will be retransmitted by the source if the source does not receive a prompt acknowledgement.

9.4.4. By far the most interesting flow control is that at the TCP-TCP level. The TCP flow control is full duplex and is the responsibility of the receiver. Control is exerted in two ways, by the window size "advertised" by the receiver and the actual rate at which TCP segments are acknowledged by the receiver. The local receive window size is sent in every T-segment header and is the best approximation of the future available buffer space for that end of the connection. It is suggestive in nature and does not guarantee that a segment of that size will actually be acknowledged if received. This suggestive nature is necessary if the full bandwidth of the connection is to be utilized. [reference 7].

9.4.5. Before continuing, we should stress that the issues of flow control, receive window control, and buffer allocation are closely coupled. It is difficult to discuss them independently. Thus comments about any of the issues must be viewed in the proper context, i.e. that window control is used to implement flow control, which is necessary to prevent overflow of available buffers.

9.4.6. While the window size is suggestive in nature, the sender *must attempt to send segments that do not exceed the sender's* send left window edge plus the foreign receive window size. Failure to do so will waste bandwidth since it is highly unlikely that the segments will be acknowledged in-full.

9.4.7. The receive window size is a function of the following:

9.4.7.1. Available user receive letter buffers

9.4.7.1.1. A window size should approximate the buffer space available at the time the window is set. This available space can be determined from the unfilled to-user buffer space in the TCB. This running total is an estimate since a to-user buffer can be returned to the user when only partially full. As will be made clear below, the window size may be more or less than the actual available space.

9.4.7.2. Previous window size

9.4.7.2.1. Smooth changes in the window size are important because they will require less frequent repackaging of segments at the sending end of the connections and avoid unnecessary shrinkage of the window to zero.

9.4.7.3. Apportionment of the bandwidth

9.4.7.3.1. Sunshine states that the bandwidth behaves almost linearly with the window size up to the maximum bandwidth. Thus if maxwindow gives maximum bandwidth to a connection then 0.1 maxwindow will yield approximately 0.1 the maximum bandwidth.

9.4.7.4. Fairness among connections

9.4.7.4.1. A certain user process may have much more buffer space available for its network connections than other user processes, perhaps because the size of the incoming letters is unpredictable or because a file transfer is in progress. This should not be used as an indication that the connection should be receiving a larger percentage of the TCP's input bandwidth. Proper window control will help administer fairness in this type of situation.

9.4.7.5. Precedence of the connection (high throughput for high precedence)

9.4.7.5.1. Higher precedence connections must be assured enough of the input bandwidth to provide the level of service set forth in the AUTODIN II specification. By establishing minimum window sizes for Category I traffic, the bandwidth is only limited by the user process's buffer space.

9.4.7.6. No attempt is made to specify what we consider the appropriate algorithm for window control since various implementation models may weigh the arguments cited above differently. In order to clarify the issues, however, an exemplary approach is presented in Appendix G.

9.4.8. Flow Control Window and Acceptability Filter Control

9.4.8.1. The receiving window is the flow control window and is suggestive of the available buffer space in the receiver. The window can vary in time, but most important, it can become smaller, perhaps retracting previous suggestions. If the flow control window is reduced such that the new right window edge (the left window edge plus the new window) falls to the left of the previous right window edge, then we say the window has shrunken from the right edge. Even though the flow control window can shrink from its right edge, the maximum right edge ever advertised defines a boundary of acceptance for the acceptability filter discussed below.

9.4.8.2. A separate sequence number range, the acceptability filter, is used to determine if a sequence number either corresponds to an old duplicate or is completely illegal. Old duplicates are always acknowledged, by sending the current receive left window edge. Illegal segments are discarded with no action. This filter can never be reduced such that its right edge is less than the maximum right edge of the flow control window previously advertised, i.e. it can never shrink from its right edge size. In addition, controls must pass this filter even if they fall to the right of its right edge.

9.4.8.3. The main motivation for the two sequence number ranges is that a flow control window of size zero, set by a receiver that wishes to stop the flow, cannot be used to detect duplicates because it could have shrunk from its right edge. Shrinking a window from the right edge at the destination essentially retracts permission already given and causes repeated discard at the destination and retransmission at the source.

9.4.8.4. Zero flow control windows

9.4.8.4.1. Setting a flow control window to zero and resetting (opening) the window are issues deferred to Appendix H.

9.4.9. Acknowledgement Generation

9.4.9.1. In general, TCP acknowledgements are innocuous and their liberal generation does not cause any particular malfunction. Nevertheless, too liberal generation of acknowledgments can cause a general reduction in overall bandwidth. [reference 8].

9.4.9.2. A good strategy seems to be to send an ACK segment only if the to-net segment queue is empty. Since solo ACKs create significant overhead, it may be profitable to piggyback ACKs on segments to retransmit as well, but this must be weighed against the cost to recompute the checksum.

9.5. Buffer Allocation Policy

9.5.1. Closely related to the issue of flow control is the buffer allocation policy. In addition to flow control considerations, proper buffer allocation should address the issues of deadlock avoidance and maximum throughput. It must be remembered that resource allocation decisions are made at two levels, the global resource level (that allocates to the user process, TCP, and SIP) and the TCP level (that allocates to the segmentizer and control handler). Even though the TCP handles buffers of the user and the SIP, it is not responsible for their allocation or their release. After using them, the TCP "returns" the buffers to the source module through the appropriate events.

9.5.2. The total picture is confused by the fact that the receiving TCP exerts flow control, which is an indication of the buffers that the user has made available, yet it has no control over the allocation of those buffers. At best it can solicit a RECEIVE from the user process, which requires no response.

9.5.3. The TCP allocation policy is thus simplified since it can only manage buffers used for segmentizing output. The following issues are of importance to the TCP space manager:

9.5.3.1. Precedence of the connection.

9.5.3.1.1. Category I activity must always have buffers available. Preemption will be used to recover resources from lower precedence connections. Of course, preemption requires that some lower precedence activity can give up a buffer.

9.5.3.2. Selectivity and discardability.

9.5.3.2.1. At heavy load it may be necessary to discard data and accept control traffic only for connections which do not have adequate user receive buffers available. This capability is necessary so that the availability of user buffers has no effect on the arrival of controls; thus their scarcity cannot result in a deadlock at the TCP connection level.

9.5.4. Preemption of Buffers

9.5.4.1. As mentioned above in section 9.2., preemption will not occur in scheduling a process; however, it will occur in all other resource management. A process responding to a high precedence event will never be denied a resource, assuming that some resource is recoverable, i.e. currently held by low precedence activity.

9.5.4.2. The algorithm for buffer preemption is made simple by the fact that within the TCP buffers are consumed only in the segmentizing of segments and produced (released) only after a segment has been acknowledged. Thus only segments on the to-net segment queue and the retransmit queue are candidates for preemption. It is fairly obvious that segments on the to-net segment queue should be selected before those on the retransmit queue. In both cases, the preemption of a segment will cause the original letter buffer to be moved from the segmentized buffer queue to the from-user buffer queue and the segmentizer will be signalled. The original sequence number range given the letter will be entered as part of the from-user buffer queue element. Note that resegmentizing of the entire letter will result in the transmission of duplicate segments, but they will be discarded gracefully at the foreign TCP.

9.6. TCP Controls

9.6.1. Controls are an essential part of any protocol. They are the "extra data" that make it possible to pass data reliably on a communication path without requiring separate control paths. TCP attempts to multiplex all its controls and data on one data path, even to the extent that controls (that require a sequence number) are assigned sequence numbers from the same sequence number space as the data.

9.6.2. Each T-segment header has a set of control bits and an independent error field. The control bits allow the piggybacking of several controls and data on one T-segment. Only one error can be present in the Control Data Extension field (see Appendix D for T-segment header details). Handling of the error field is simple, The errors take up no sequence numbers and receive no acknowledgement. Listed below are the control bits of the T-segment header, which define the controls that can be passed between TCP's. Those that consume sequence number space are marked with "*".

SYN: *Request to synchronize the send sequence numbers.

ACK: There is a valid Acknowledgement in the ACK field.

FIN: *Sender will stop SENDing and RECEIVEing on this connection.

DSN: *The sender has stopped using sequence numbers and wants to initiate a new sequence number for sending.

EOL: This segment is the last segment of a letter.

FL: *The sender wants network-to-user data on the foreign end of this connection to be flushed immediately.

INT: *The sender wants to INTERRUPT on this connection.

NOP: *A no action segment that uses sequence number space and requires an ACK.

WOPEN: *The sender has opened a previously zero receive window.

WACK: *The sender wishes to acknowledge the corresponding WOPEN.

9.6.3. Since several controls may be present in one T-segment, random handling of controls can produce unexpected behavior. It is necessary that implementers of the TCP protocol process controls uniformly. Part of the handling is defined by the state diagram, Figure 7, and the set of transitions listed in Appendix F. However, it is also necessary to define the handling of controls with respect to sequence and priority. Below, the assignment of sequence numbers to received controls and the control processing order are discussed.

9.6.3.1. Received controls (in the same T-segment header) should be assigned sequence numbers as follows:

9.6.3.1.1. SYN, FL, INT, text, DSN, FIN

9.6.3.1.2. Note that WOPEN and WACK use a sequence number, but they can never be piggybacked with either data or controls, and the sequence number is not consumed.

9.6.3.2. The processing order for controls arriving in the same T-segment or for controls queued to be processed is dependent on the state of the connection. For the ESTD state, the order is the following:

9.6.3.2.1. SYN, ACK, FL, INT, text, DSN, FIN, NOP

9.6.4. Some combinations of controls are especially meaningful for a connection. Examples of these are the INT-FL and the FIN-FL pairs, which represent an interrupt with flushing and an immediate close, respectively. Below is more description on the meaning INT's and FL's.

9.6.4.1. It is expected that the interrupt control (INT) will often be accompanied by the flush control (FL), in which case the actions associated with both are carried out.

9.6.4.2. The interrupt control may be sent with any segment. When a TCP receives it, the destination process is immediately signalled (by whatever mechanism is available) that an interrupt has occurred. The interrupt control occupies a sequence number and therefore can be synchronized with the data.

9.6.4.3. The flush control may be sent with any segment. Upon receipt of a FL, all data with lower sequence numbers than the flush signal is discarded but acknowledged as if it were processed. An event notifying of the discarding of data is sent to the receiving user process.

9.6.4.4. The effect of a flush is that the receive left window edge may be set to the sequence number of the flush control, and an acknowledgement may be sent.

9.6.4.5. It is necessary for the TCP to follow two rules with respect to flushing. First, a FL , INT-FL, or FIN-FL should not be sent while in the process of resynchronizing. Second, the WOPEN control should be resent using a new sequence number if it is flushed.

9.7. Error Handling

9.7.1. Error handling and recovery consists of transmitting and receiving either Error or RESET segments.

9.7.2. Error segments are segments that have "001" in the control dispatch field, occupy no sequence number space, and have a sequence number equal to the left send window edge (if there is one). Error segments have the error type in the Control Data Extension field of the T-segment header. The error types are a subset of all the possible event codes, described in Appendix B, Event Codes. The Error segments that can be transmitted, along with the conditions under which they are generated follow:

EFP 6 - unacceptable SYN [SYN/ACK]

generated during recovery from half-open connections. (see 9.1.2.3.)

EFP 7 - connection non-existent

generated during address check at foreign TCP

EFT 8 - foreign TCP inaccessible

generated by a busy foreign TCP (never sent on a Category I connection, unless all connections at that TCP are currently Category I)

EFP 15 - security violation at foreign TCP

generated during authorization table lookup at foreign TCP

EFP 16 - precedence violation at foreign TCP

generated during precedence consistency check at foreign TCP

EFP 17 - TCC violation at foreign TCP

generated during authorization table lookup at foreign TCP

RESET - foreign TCP has reset

generated only if TCP is in the SYNsent state and gets an EFP 6 that is believable.

9.7.3. The sequence number of the segment that caused the error is placed in the ACK field of the Error or RESET segment. In the case of the RESET segment, the ACK field contains the sequence number of the Error segment that caused the RESET's generation. This is in order that the segments can be believed when they are received, and not confused with duplicates. The ACK bit is always set in Error or RESET segments. Error and RESET segments are neither acknowledged (with an ACK) nor retransmitted.

9.7.4. The errors are generated as follows:

9.7.5. Errors 7 and 8 are very simple to understand and implement. First consider the arrival of either segment. If the state of the connection is in any of the synchronizing states then the state is changed to OPEN and the synchronization efforts resumed from scratch. If this repeatedly happens for a CONNECT attempt, a synchronization timer will timeout and the connection will be closed. If the state of the connection is in any of the FIN processing states then the TCP will simulate (to the user) that the connection was closed with a FL-FIN from the foreign TCP. If the connection state is ESTD then the user is given an error message. The message implies that something is amiss, and the user can carry on (if the message was EFT 8) or can close the connection.

9.7.6. Errors 15, 16, and 17 are fatal errors and will usually cause the connection to be closed. The sender of one of these errors moves to the CLOSED state and at worst leaves a half-open connection (in the unlikely event that the error does not reach the destination). The receiver of one of these errors usually simulates a flushing close. The exception to this is the receipt of Errors 16 and 17 during the synchronization states, which cause a transition to the OPEN state. Thus an S/P/T violation is severely handled and forces the violating user into the CLOSED state. If the user wishes to attempt another connection, he will have to reopen the connection manually.

9.7.7. Error EFP 6 and RESET are for the purpose of establishing connections reliably, and for being able to recover from crashes or loss in synchronization. Loss in synchronization may occur owing to long delays and lost segments. In short this mechanism recovers from the existence of half open connections.

9.7.8. A RESET is ONLY generated if the TCB is in SYNsent state, and gets Error EFP 6. The receipt of this error (when trying to open a connection) implies at the worst that the other end is in a half open i.e. established state. Hence it must be reset.

9.7.9. Believability Test

9.7.9.1. When testing to see if an Error packet is believable it may be sufficient to check if the ACK field refers to something we sent that has not been ACKed. To believe a RESET packet we also have to check that not only does the ACK refer to something we sent, but we also did in fact send an EFP 6. This additional precaution is necessary since the consequence of getting a RESET are grave. A malicious TCP may fabricate a RESET which ACKs something that the destination TCP has sent. If that TCP believes the RESET without making sure it really sent an EFP 6 then we may just have lost a connection. It is true that malicious TCP's could cause havoc, but it seems worth making redundant checks where appropriate. It does not seem that giving Error or RESET control a sequence number of their own will solve any of the problems we have.

10. HIGHER-LEVEL PROTOCOLS

10.1. It is envisioned that the TCP will be able to support higher level protocols efficiently. It should be relatively easy to interface existing ARPANET-like protocols to the TCP. The S/P/T and terminal-to-terminal requirements of the AUTODIN II network will require additional work in these areas, however. An example of the types of protocols that will be useful are a Telnet-like protocol and a file transfer protocol.

10.2. At some point, a set of well known sockets must be selected that will provide consistent access to the functions provided by the server side of the processes that implement the higher level protocol. These sockets must have a constant port identifier so that only the host identifier (TCP identifier) changes in any well known socket.

11. MEASUREMENTS

11.1. To evaluate the extent to which the TCP and PSN can deliver services as required and to identify the dominant parameters that affect that service, a comprehensive set of performance measurements must be made. Measurement will also be a valuable tool for determining the initial settings for the values of TCP parameters.

11.2. Measurement Implementation Philosophy

11.2.1. We view the measurement process as something which occurs internal to the TCP but which is controllable from outside. A well known socket owned by the TCP can be used to accept control which will select one or more measurement classes to be collected. The data would be periodically sent to a designated foreign socket which would absorb the data for later processing, in the manner currently used in the ARPANET IMPs. Each measurement class has its own data segment format to make the job of parsing and analyzing the data easier.

11.2.2. We would restrict access to TCP measurement control to a few designated sites. This is easily done by setting up listening control connections on partially specified foreign sockets.

11.3. Performance Measurement

11.3.1. Throughput and delay should be measured as a function of precedence level, maximum segment size, window shrinking algorithms, buffer allocation strategies, retransmission rates, fragmentation, letter sizes, etc. Overhead should be evaluated in several modes of user activity.

11.4. Single Connection Measurements

11.4.1. Round Trip Delay Times

11.4.1.1. Time from moment the segment is sent by the TCP to the time that the ACK is received by the TCP.

11.4.1.2. Time from the moment the user issues the SEND to the time that the user gets the successful return code.

11.4.1.2.1. Note: Segment size should be used to distinguish from one set of round trip times and another.

11.4.1.2.2. Network destination, and current configuration and traffic load may also be issues of importance that must be taken into account.

11.4.1.2.3. The histogram of round trip times include retransmission times and these must be taken into account in the analysis and evaluation of the collected data.

11.4.2. Segment Size Statistics

11.4.2.1. Histogram of segment length in both directions on the full duplex connection.

11.4.2.2. Histogram of letter size in both directions.

11.4.3. Measure of Disorderly Arrival

11.4.3.1. Distance from the first octet of arriving segment to the left window edge. A histogram of this measure gives an idea of the out of order nature of segment arrivals. It will be 0 for segments arriving in order.

11.4.4. Retransmission Histogram

11.4.5. Effective Throughput

11.4.5.1. This is the effective rate at which the left edge of the window advances. The time interval over which the measure is made is a parameter of the measurement experiment. The shorter the interval, the more bursty we would expect the measure to be.

11.4.5.2. It is possible to measure effective data throughput in both directions from one TCP by observing the rate at which the left window edge is moving on ACK sent and received for the two windows.

11.4.5.3. Since throughput is largely dependent upon buffer allocation and window size, we must record these values also. Varying window for a fixed file transmission might be a good way to discover the sensitivity of throughput to window size.

11.4.6. Output Measurement

11.4.6.1. The throughput measurement is for data only, but includes retransmission. The output rate should include all octets transmitted and will give a measure of retransmission overhead. Output rate also includes segment format overhead octets as well as data.

11.4.7. Utilization

11.4.7.1. The effective throughput divided by the output rate gives a measure of utilization of the communication connection.

11.4.8. Window and Buffer Allocation Measurements

11.4.8.1. Histogram of letters outstanding, measured at the instant of SEND receipt by TCP from user or at instant of arrival of a letter for a receiving user.

11.4.8.2. Buffers in use on the SEND side upon segment departure into the net; buffers in use on the RECEIVE side upon delivery of a segment into a USER buffer.

11.5. Multiconnection Measurements

11.5.1. Statistics on User Commands sent to the local TCP

11.5.2. Statistics of error or success codes returned [histogram of each type of error or return response]

11.5.3. Statistics of control bit use

11.5.3.1. Counter for each control bit over all segments emitted by the TCP and another for segments accepted

11.5.4. Count data carrying segments

11.5.5. Count ACK segments with no data

11.5.6. Error segments distribution by error type code received from the net and sent out into the net

11.6. TCP Exerciser

11.6.1. Early implementers of TCP's on the ARPANET have found an exerciser program useful. It exercises a given TCP, causing it to cycle through a number of states; opening, closing, and transmitting on a variety of connections. This program collects statistics and generally tries to detect deviation from TCP functional specifications. Clearly there must be a copy of this program both at the local site being tested and some site which has a certified TCP. There must be a master-slave relationship so the master can tell the slave what's going wrong with the test.

11.7. Minimum Measurement Facilities

11.7.1. The correct algorithms and parameter settings for flow control and buffer allocation must evolve from experience with TCP implementations. Thus it is suggested that some measurement facilities in TCP remain through initial implementation and acceptance testing and into the "production" phase. In fact, it may be that in some instances the measurement facilities will be used to alter critical parameters in real-time. Therefore, the following measurement facilities should be present in all multiple connection TCP's:

11.7.2. Effective Throughput

11.7.2.1. This is the effective rate at which the left edge of the window advances. The time interval over which the measure is made is a parameter of the measurement experiment. The shorter the interval, the more bursty we would expect the measure to be.

11.7.2.2. It is possible to measure effective data throughput in both directions from one TCP by observing the rate at which the left window edge is moving on ACK sent and received for the two windows.

11.7.3. Retransmission Rate

11.7.3.1. The retransmissions per segment is an indication of how well the foreign TCP is living up to its advertised segment consumption rate. It will be valuable for setting send window sizes.

11.8. Error Log

11.8.1. The TCP should log in a file the protocol errors it detects. The error records should include a time stamp, the TCP header, and the TCB contents at the time of the error. It might be desirable to also include the data portion of the segment in error.

11.8.2. This log should be examined daily by the computer operator or chief programmer, and errors due to local problems corrected. Errors due to remote problems should be reported to the remote site and to the NCC.

11.8.3. It is especially important to record violations of S/P/T in this error log (or another log). S/P/T violations should be reported to the NCC periodically.

11.9. Debugging Aids

11.9.1. When TCP implementations are being tested, either initially or after a modification or parameter change, it is desirable to track the execution of the TCP in some detail. In particular it is useful to log the headers or even the full segments for some or all of the messages sent and received by the TCP. The TCP should be prepared to log such information when a debugging flag is set.

12. REFERENCES

(1) Defense Communications Agency, "Specifications (Type A) for AUTODIN II Phase I," November 1975 (referred to as the AUTODIN II Specification).

(2) R. Tomlinson, "Selecting Sequence Numbers," INWG Protocol Note #2, September 1974.

(3) Y. Dalal, "More on Selecting Sequence Numbers," INWG Protocol Note #4, October 1974.

(4) V. Cerf, Y. Dalal, C. Sunshine, "Specification of Internet Transmission Control Program," INWG General Note #72, December 1974 (Revised).

(5) Cerf, V., "TCP Resynchronization," SU-DSL Technical Note #79, January 1976.

(6) Cerf, V. and R. Kahn, "A Protocol for Packet Network Intercommunication," IEEE Transactions on Communication, Vol COM-20, No. 5, May 1974.

(7) C. Sunshine, "Interprocess Communication Protocols for Computer Networks," Digital Systems Laboratory Technical Note #105, December 1975.

(8) Cerf, V., "ARPA Internetwork Protocols Project Status Report for the Period November 15, 1975 - February 15, 1976", Digital Systems Laboratory Technical Note 83, Stanford University, Stanford, California, 25 February 1976.

(9) Y. Dalal, "Establishing a Connection," INWG Protocol Note #14, March 1975.

STANFORD RESEARCH INSTITUTE
Menlo Park, California 94025 · U S A

Transmission Control Protocol Specification
Appendices

15-July-76

Jonathan B. Postel
Larry L. Garlick
Raphael Rom

Augmentation Research Center

Stanford Research Institute
Menlo Park, California 94025

(415) 326-6200

This report is the deliverable "Transmission Control Protocol Specification (Final)" as specified in contract DCA100-76-C-0034.

APPENDIX A. ADDRESSING

A. Addressing is a general mechanism for routing messages (text, letters, segments, packets, ...) to a destination. The addressing requirements of an each entity (user, THP, TCP, PS, ...) may be quite different since the destination entity is almost always different. In the AUTODIN II network, several layers of addressing exist. These layers are kept as invisible as possible, in hopes of presenting one coherent addressing model to the user. Below, a few of the more important layers of addressing are discussed, in order of proximity to the terminal user.

B. User Addressing

1. A user address must be simple enough to permit destinations to be specified by Subscriber ID, when the user is supported by a TAC. Yet also it must be powerful enough to address an experimental version of a standard function on some host computer. To this end, a user address has the following form:

Subscriber ID. (16 bits) This field corresponds to a terminal user on a TAC or to a host-entity. It is a required part of the address.

User ID. 12 bits) This field is used for addressing host users or standard host functions and consists of two subfields as follows:

User ID Type FLag. (1 bit) A zero means the User ID Proper is a static ID, such as directory name; a one means the User ID Proper is dynamic, such as a process name or job number.

User ID Proper. (11 bits) Depending on the previous bit, this field may contain a static or dynamic ID. The field will be zero for a TAC subscriber.

Function Suffix. (4 bits) This field is used to select a particular function offered by the user above. It defaults to zero, suggesting that a user's well known address should have a zero Function Suffix.

2. Examples

User B is a terminal user supported by a TAC and always listens for a connection attempt on his default Function Suffix. The address of user B is <subscriber ID = B> <user ID = 0> <Function Suffix = 0>. This is the simplest address that can be specified.

Suppose host D has a special process, C, that provides an experimental version of the standard function service, E. To access this service, an address like the following will be used:

<subscriber = D> <userid = C's static ID> <Function Suffix = E>

This represents one of the most complex types of user addressing offered. It demonstrates the use of a static User ID to provide a "well-known" address for a global addressing mechanism.

C. TCP Addressing

1. The TCP addressing scheme is based on sockets. A socket is the concatenation of a TCP identifier and a port, each of which is a 16 bit entity. The conversion of user addresses to TCP addresses is accomplished using the following rules:

(1) The Subscriber ID becomes the TCP identifier if a User ID is provided, i.e. if the Subscriber ID refers to a host.

(2) The User ID, if present, maps directly into the leftmost 12 bits of the port.

(3) The Function Suffix, if present, maps directly into the rightmost 4 bits of the port.

2. The result of these rules is that a TCP identifier is nothing more than a Subscriber ID and is at least enough to route a message to a TCP. In the TAC case, the TCP is able to route the message to the user (end subscriber) with the extra leader added by the PS. Routing messages in the host case, however, requires port information.

D. BSL Address

1. The destination address found in the Binary Segment Leader (BSL) always corresponds exactly to a subscriber address. This address is required and will be taken directly from the address provided by the user. In the TAC case, the address refers to a terminal connected to an access circuit. The destination PS will handle routing to the TAC and the adding of any leaders required for final routing to the user terminal. For the host case, the address is that of the host's access circuit and cannot be used to address specific users or standard host functions.

APPENDIX B. EVENTS

A. The TCP communicates with the SIP and the user process through event sending. Below are the set of events that are passed to/from the TCP at each interface.

B. TCP - SIP

1. DATA (TCP <=> SIP)

```
!---------------------------------!
!            OP: DATA             !
!---------------------------------!
!         Subscriber ID           !
!---------------------------------!
!         T-segment ID            !
!---------------------------------!      !---------------!
!         Block pointer  ---------!----->! segment leader !
!---------------------------------!      !---------------!
                                         !  TCP header   !
                                         !---------------!
                                         !   T-segment   !
                                         !     text      !
                                         !---------------!
```

2. ERROR (SIP => TCP only)

```
!---------------------------------!
!            OP: ERROR            !
!---------------------------------!
!         Subscriber ID           !
!---------------------------------!
!         T-segment ID            !
!---------------------------------!
!            reason               !
!---------------------------------!
```

Possible reason

Reject by PS

security violation (at src)
security violacion (at dst)
illegal precedence (at src)
illegal precedence (at dst)

illegal user group
illegal address
flow control

Incomplete transmission

lost in net

Undelivered

host dead
host busy
wrong precedence

Other errors

without ID
with ID
PS going down

3. STOP/RESUME transmission (TCP => SIP only)

```
!--------------------------------!
!      OP: general control       !
!--------------------------------!
!         Subscriber ID          !
!--------------------------------!
!        T-segment ID = 0        !
!--------------------------------!
!       reason: stop/resume      !
!--------------------------------!
!      type: src addr/ TCC       !
!--------------------------------!
!        value:  see type        !
!--------------------------------!
```

4. Going Inoperable (TCP => SIP only)

```
!---------------------------------!
!      OP: general control        !
!---------------------------------!
!          Subscriber ID          !
!---------------------------------!
!         T-segment ID = 0        !
!---------------------------------!
!       reason: going down        !
!---------------------------------!
!            how soon             !
!---------------------------------!
!            how long             !
!---------------------------------!
!           description           !
!---------------------------------!
```

Possible description

Scheduled maintenance
Panic
Reload
Emergency restart

5. ACK (TCP <=> SIP)

```
!---------------------------------!
!      OP: specific control       !
!---------------------------------!
!          Subscriber ID          !
!---------------------------------!
!           T-segment ID          !
!---------------------------------!
!   reason: args/xmission ctl     !
!---------------------------------!
```

6. Negative ACK (TCP <=> SIP)

```
!--------------------------------!
!      OP: specific control      !
!--------------------------------!
!         Subscriber ID          !
!--------------------------------!
!          T-segment ID          !
!--------------------------------!
!      reason: args NOT OK       !
!--------------------------------!
```

C. TCP - THP (or other process)

1. OPEN

a. USER => TCP

```
!--------------------------------!
!            OP: OPEN            !
!--------------------------------!
!          Subscriber ID         !
!--------------------------------!
!         Transaction ID         !
!--------------------------------!
!           Local port           !
!--------------------------------!
!      [Foreign socket list]     !
!--------------------------------!
!     [Send S / P / TCC list]    !
!--------------------------------!
!  [Minimum Receive Precedence]  !
!--------------------------------!
```

b. TCP => USER

```
!--------------------------------!
!            OP: OPEN            !
!--------------------------------!
!          Subscriber ID         !
!--------------------------------!
!         Transaction ID         !
!--------------------------------!
!      Local connection name     !
!--------------------------------!
!            S/P/TCC             !
!--------------------------------!
!          description           !
!--------------------------------!
```

Possible description

See "Error and TCP-to-User Event Codes."

2. SEND

a. USER => TCP

```
!--------------------------------!
!            OP: SEND            !
!--------------------------------!
!         Subscriber ID          !
!--------------------------------!
!         Transaction ID         !
!--------------------------------!
!     Local connection name      !
!--------------------------------!
!         Buffer address         !
!--------------------------------!
!           Byte count           !
!--------------------------------!
!            EOL flag            !
!--------------------------------!
!            Security            !
!--------------------------------!
```

b. TCP => USER

```
!--------------------------------!
!            OP: SEND            !
!--------------------------------!
!         Subscriber ID          !
!--------------------------------!
!         Transaction ID         !
!--------------------------------!
!     Local connection name      !
!--------------------------------!
!         Buffer address         !
!--------------------------------!
!          description           !
!--------------------------------!
```

Possible description

See "Error and TCP-to-User Event Codes."

3. RECEIVE

a. USER => TCP

```
!--------------------------------!
!           OP: RECEIVE          !
!--------------------------------!
!         Subscriber ID          !
!--------------------------------!
!         Transaction ID         !
!--------------------------------!
!     Local connection name      !
!--------------------------------!
!         Buffer address         !
!--------------------------------!
!           Byte count           !
!--------------------------------!
!      Mode:  Full/partial       !
!--------------------------------!
```

b. TCP => USER

```
!--------------------------------!
!           OP: RECEIVE          !
!--------------------------------!
!         Subscriber ID          !
!--------------------------------!
!         Transaction ID         !
!--------------------------------!
!     Local connection name      !
!--------------------------------!
!         Buffer address         !
!--------------------------------!
!           Byte count           !
!--------------------------------!
!            EOL flag            !
!--------------------------------!
!            Security            !
!--------------------------------!
!          description           !
!--------------------------------!
```

Possible description

See "Error and TCP-to-User Event Codes."

4. RETRACT (all receive buffers)

   a. THP => TCP

```
!--------------------------------!
!          OP: RETRACT           !
!--------------------------------!
!         Subscriber ID          !
!--------------------------------!
!         Transaction ID         !
!--------------------------------!
!     Local Connection Name      !
!--------------------------------!
```

   b. TCP => THP

```
!--------------------------------!
!          OP: RETRACT           !
!--------------------------------!
!         Subscriber ID          !
!--------------------------------!
!         Transaction ID         !
!--------------------------------!
!     Local Connection Name      !
!--------------------------------!
!    Current Buffer Address      !
!--------------------------------!
!       Bytes Transferred        !
!--------------------------------!
! Description:   OK / Not found  !
!--------------------------------!
```

5. CLOSE

   a. USER => TCP

```
!--------------------------------!
!           OP: CLOSE            !
!--------------------------------!
!         Subscriber ID          !
!--------------------------------!
!         Transaction ID         !
!--------------------------------!
!     Local connection name      !
!--------------------------------!
!  Type:  immediate / deferred   !
!--------------------------------!
```

b. TCP => USER

```
!--------------------------------!
!           OP: CLOSE            !
!--------------------------------!
!         Subscriber ID          !
!--------------------------------!
!         Transaction ID         !
!--------------------------------!
!     Local connection name      !
!--------------------------------!
!          description           !
!--------------------------------!
```

Possible description

See "Error and TCP-to-User Event Codes."

6. INTERRUPT

a. USER => TCP

```
!--------------------------------!
!         OP: INTERRUPT          !
!--------------------------------!
!         Subscriber ID          !
!--------------------------------!
!         Transaction ID         !
!--------------------------------!
!     Local connection name      !
!--------------------------------!
!     Type:  flush / no flush    !
!--------------------------------!
```

b. TCP => USER

```
!--------------------------------!
!          OP: INTERRUPT         !
!--------------------------------!
!          Subscriber ID         !
!--------------------------------!
!         Transaction ID         !
!--------------------------------!
!     Local connection name      !
!--------------------------------!
!          Description           !
!--------------------------------!
```

Possible description

See "Error and TCP-to-User Event Codes."

7. FLUSH

a. USER => TCP

```
!--------------------------------!
!           OP: FLUSH            !
!--------------------------------!
!          Subscriber ID         !
!--------------------------------!
!         Transaction ID         !
!--------------------------------!
!     Local connection name      !
!--------------------------------!
```

b. TCP => USER

```
!--------------------------------!
!          OP: FLUSH             !
!--------------------------------!
!        Subscriber ID           !
!--------------------------------!
!        Transaction ID          !
!--------------------------------!
!     Local connection name      !
!--------------------------------!
!         Description            !
!--------------------------------!
```

Possible description

See "Error and TCP-to-User Event Codes."

8. STATUS

a. USER => TCP

```
!--------------------------------!
!          OP: STATUS            !
!--------------------------------!
!        Subscriber ID           !
!--------------------------------!
!        Transaction ID          !
!--------------------------------!
!     Local connection name      !
!--------------------------------!
```

b. TCP => USER

```
!--------------------------------!
!           OP: STATUS           !
!--------------------------------!
!         Subscriber ID          !
!--------------------------------!
!         Transaction ID         !
!--------------------------------!
!     Local connection name      !
!--------------------------------!
!          Description           !
!--------------------------------!
!     Status block pointer --!   !
!----------------------------!---!
                             !
                        !----!------------!
                        !    TCB status   !
                        !-----------------!
```

9. GENERAL

a. USER => TCP

```
!--------------------------------!
!           OP: GENERAL          !
!--------------------------------!
!         Subscriber ID          !
!--------------------------------!
!       Transaction ID = 0       !
!--------------------------------!
!       reason: going down       !
!--------------------------------!
!            how soon            !
!--------------------------------!
!            how long            !
!--------------------------------!
!          description           !
!--------------------------------!
```

Possible description

Scheduled maintenance
Panic
Reload
Emergency restart

b. TCP => USER

```
!--------------------------------!
!          OP: GENERAL           !
!--------------------------------!
!         Subscriber ID          !
!--------------------------------!
!         Transaction ID         !
!--------------------------------!
!     Local connection name      !
!--------------------------------!
!          Description           !
!--------------------------------!
```

Possible description

See "Error and TCP-to-User Event Codes."

10. STOP TRANSMISSION

a. USER => TCP

```
!--------------------------------!
!      OP: STOP TRANSMISSION     !
!--------------------------------!
!         Subscriber ID          !
!--------------------------------!
!      Transaction ID = 0        !
!--------------------------------!
!     Type:  src addr/TCC        !
!--------------------------------!
```

b. TCP => USER

None.

11. RESUME TRANSMISSION

a. USER => TCP

```
!-------------------------------!
!    OP: RESUME TRANSMISSION    !
!-------------------------------!
!         Subscriber ID         !
!-------------------------------!
!      Transaction ID = 0       !
!-------------------------------!
!    Type:  src addr/TCC        !
!-------------------------------!
```

b. TCP => USER

None.

12. MOVE CONNECTION

a. USER => TCP

```
!-------------------------------!
!    OP: MOVE CONNECTION        !
!-------------------------------!
!         Subscriber ID         !
!-------------------------------!
!      Transaction ID = 0       !
!-------------------------------!
!        New process ID         !
!-------------------------------!
```

b. TCP => USER

None.

13. ACK / NACK

a. THP <=> TCP

```
!-----------------------------------!
!              OP: ACK              !
!-----------------------------------!
!           Subscriber ID           !
!-----------------------------------!
!          Transaction ID           !
!-----------------------------------!
! Reason: args OK / args not OK     !
!-----------------------------------!
```

D. Error and TCP-to-User Event Codes

1. The event code specifies the particular event that the TCP wishes to communicate to the user or foreign TCP.

2. In addition to the event code, three flags may be useful to classify the event into major categories and facilitate event processing by the user:

E flag: set if event is an error

L/F flag: indicates whether event was generated by Local TCP, or Foreign TCP or network

P/T flag: indicates whether the event is Permanent or Temporary [retry may succeed]

3. Events are encoded into 8 bits with the high order bits set to indicate the state of the E, L/F, and P/T flags, respectively

4. Events specified so far are listed below with their codes and flag settings. A * means a flag does not apply or can take both values for this event. Additional events may be defined as needed.

0	0**	general success
1	ELP	connection illegal for this process
2	OF*	unspecified foreign socket has become bound
3	ELP	connection not open
4	ELT	no room for TCB
5	ELT	foreign socket unspecified
6	ELP	connection already open
	EFP	unacceptable SYN [or SYN/ACK] arrived at foreign TCP. (Note: This is not a misprint, the local meaning is different from foreign.)
7	EFP	connection does not exist at foreign TCP
8	EFT	foreign TCP inaccessible (may have subcases)
9	ELT	retransmission timeout

10	E*P	buffer returned due to flush
11	OF*	interrupt to user
12	**P	connection closing
13	E**	general error
14	E*P	connection reset
15	E*P	security violation
16	E*P	illegal precedence
17	E*P	user group (TCC) violation
18	EFP	stop transmission at destination
19	E*P	buffer flushed due to stop transmission
20	OL*	solicit RECEIVE from user
21	E*P	buffer flushed due to preemption
22	ELP	synchronization timeout
23	OLT	remote user not receiving

5. Possible events for each message type are as follows:

Type 0[general]: 2,11,12,14,20

Type 1[open]: 0,1,4,6,13,15,16,17,22

Type 2[close]: 0,1,3,13

Type 3[interrupt]: 0,1,3,5,7,8,9,12 13,15,16,17,18

Type 10[send]: 0,1,3,5,7,8,9,10,11,12,13,15,16,17,18,19,21,23

Type 20[receive]: 0,1,3,10,12,13,15,16,17,18,19,21

Type 30[status]: 0,1,13

6. Note that events 6(foreign), 7, 8, and 15 - 18 are generated at the foreign TCP or in the network, and these same codes are used in the Control Data Extension field of the T-segment header.

APPENDIX C. BINARY SEGMENT LEADER

WORD	BITS	DEFINITION
+0	1-5	Precedence Field. Bit 1 is for a Category I redundancy check. Bits 2-5 provide 16 levels of precedence
+0	6-12	Spare.
+0	13-16	Security Field. This field allows up to 16 security designators.
+1	1-8	Transmission Control Codes. Provides a means to compartmentalize traffic and define controlled communities of interest among subscribers.
+1	9-10	Spare.
+1	11-16	Command Control Code. Identifies the segment content as data, service message, etc.
+2	1-4	Redundant Security Field. This field has a different bit orientation within the machine word than the security field, and it provides a means to detect store/fetch malfunctions if the hardware does not have bus parity checks.
+2	5-12	Redundant TCC. Provides the same malfunction protection for TCC as provided for security.
+2	13-16	Spare.
+3	1-16	Destination logical address.
+4	1-4	TCP Version Number.
+4	5-16	Segment Number.

For a graphical representation of the BSL and its relationship to headers required by higher level protocols, see Figure D1.

APPENDIX D. TCP (T-SEGMENT) HEADER

The header below supports all the functions and implementation suggestions mentioned in the body of the specification. In addition, it contains fields for internetworking, labelled "[I-net]," even though no mention of internetworking has been made in the specification. Following the header is a section discussing the issues related to reducing the header size. A graphical representation of the T-segment header and its relationship to the headers required by other levels of protocols can be seen in Figure D1.

4 bits: Protocol version information

8 bits: Header length in octets

12 bits: Length of text in octets

8 bits: Padding (all zeroes)

32 bits: Sequence number of first control or data octet

```
16 bits: Control Information
  Listed from most to least significant bits:
  SYN: Request to synchronize the send sequence numbers
  ACK: There is a valid Acknowledgement in the 32 bit ACK field
  FIN: Sender will stop SENDing and RECEIVEing on this connection
  DSN: The sender has stopped using sequence numbers and wants to
       initiate a new sequence number for sending.
  EOL: This segment is the last segment of a letter.
  FL:  The sender wants network-to-user data on the foreign end of
       this connection to be flushed immediately.
  INT: The sender wants to INTERRUPT on this connection.
  NOP: A no action segment that uses sequence number space and
       requires an ACK.
  WOPEN: The sender has opened a previously zero receive window.
  WACK: The sender wishes to acknowledge the corresponding WOPEN.
  BOS:  This segment includes the beginning a T-segment. [I-net]
  EOS:  This segment includes the end a T-segment. [I-net]
  XXX:  one (1) unused control bits
  CD: three (3) bits of control dispatch:
    000: Null (the control octet contents should be ignored)
    001: Event Code is present in the control octet. (see Appendix
         A)
    010: Special Functions
    011, 1XX: Unused
```

16 bits: Destination port address

16 bits: Source port address

8 bits: Destination network address [I-net]

16 bits: Destination TCP address [I-net]

8 bits: Source network address [I-net]

16 bits: Source TCP address [I-net]

32 bits: Acknowledgement number (i.e. sequence number of next octet expected).

16 bits: Window size (in octets)

8 bits: Control Data Extension
If CD is 000 then this octet is to be ignored.
If CD is 001, this octet contains event codes defined in Appendix B.
If CD is 010, this octet contains a special function code as defined below:
0: RESET the specific connection referenced in this segment
1: QUERY status of connection referenced in this segment
2: STATUS Reply to QUERY with requested status.
>2: Unused
If CD is 011 or 1XX, this octet is undefined (all zeros).

16 bits: Checksum

PROTOCOL HEADERS

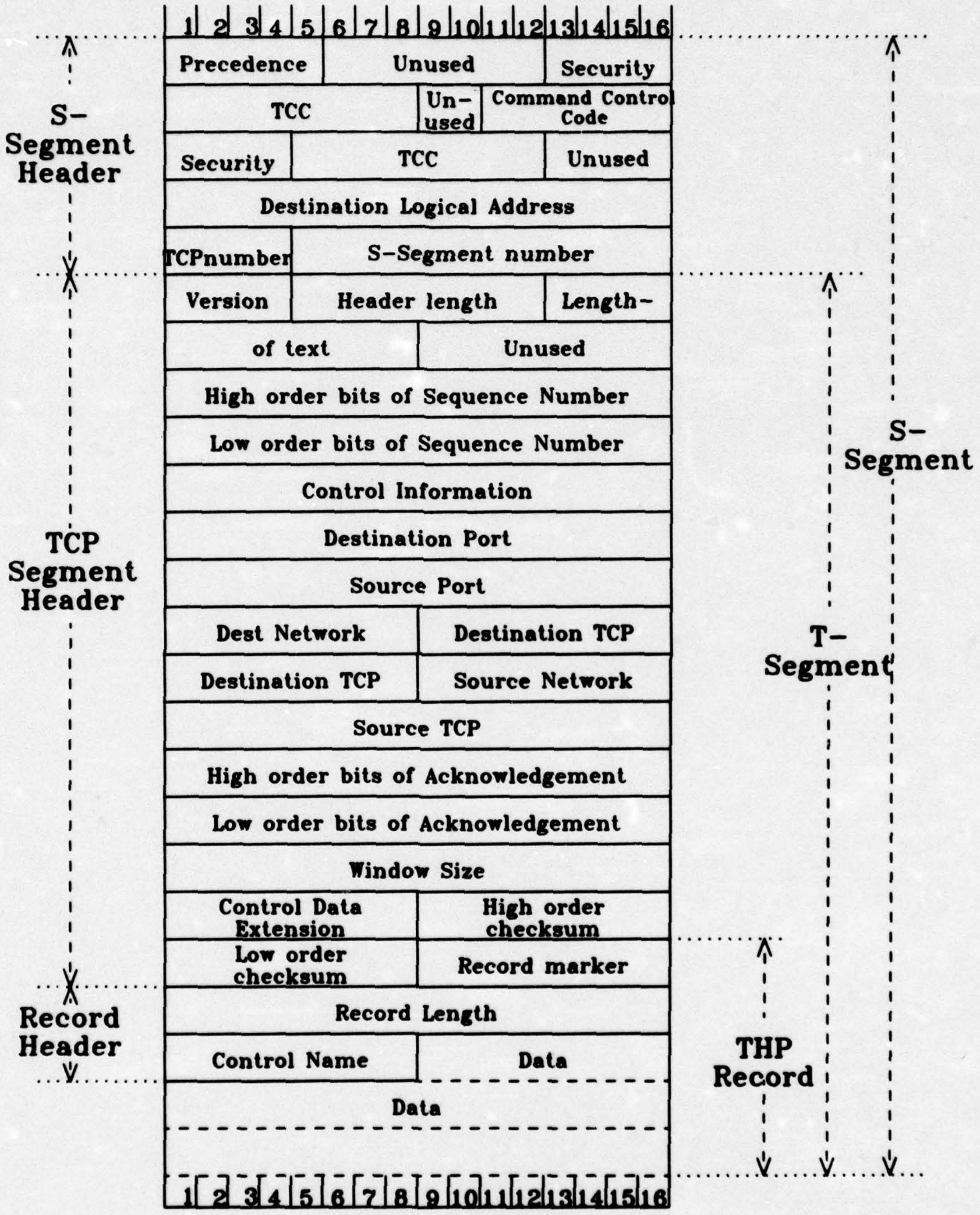


Figure D1

Header Minimization Issues

Above we presented a version of the TCP header that we believe to be complete with respect to capabilities and generous in terms of providing adequate space for growth. Now we address the issues regarding certain tradeoffs that can be made between the cost savings of a smaller header on one hand, and restricted capabilities, expensive implementation, and reliability problems on the other.

The main motivation for reducing the size of the TCP header is to decrease the overhead associated with the transmission of each data or control segment. Since a T-segment header is present in every S-segment sent or received by a TCP, a reduced header could represent an increase in bandwidth.

Unless fields can be identified that are redundant, unsuitable for the network, or unnecessarily large, a reduction in header sizes may be accompanied by an increase in the difficulty of segment handling. This may be evident in more complex segment header parsing or perhaps in the increase of buffer resource consumption.

The T-Segment header presented earlier is conservative with respect to potential growth of the network and internetting considerations. The large sequence number space represents the concern that the maximum segment lifetime and TCP bandwidth will increase. While no mention of internetting is made in the body of the specification, the internetting fields have been included so that the capability can be easily added without altering the header.

Sequence number space reduction

Reducing the sequence number space directly reduces the size of the sequence number field and the acknowledgement field. The size indicated in the TCP header is sufficient to handle the internetwork case and may appear conservative if internetting is not a requirement of the network. Below we discuss several issues related to the size of the sequence number space--byte size, maximum segment lifetime, and resynchronization.

Byte size issues

This specification uses the octet (8 bits) as the byte size or the unit to which sequence numbers are assigned. A potential savings in TCP header is the change to a larger byte size, so that the sequence number space and the corresponding sequence number, acknowledgement, and window size fields could be reduced. (Notice that a change in the byte size does not necessarily change the segment size field, since this would require padding in the sender to fill out the final byte of a segment.)

The major concern is that an increase in the byte size will increase the unit of flow control, requiring the receiver to buffer at least a full byte each time a non-zero window size is offered. Since segments can only be acknowledged on sequence number boundaries, partial acknowledgement of bytes would not be possible.

The byte size must be chosen carefully, especially with respect to internetting. The byte size must never exceed the minimum unit of delivery within any network that may pass the segment. Fragmentation at network gateways must break the segment at byte boundaries.

Finally, as shown below, savings by reducing the byte size are realized logarithmically with the inverse of the byte size, i.e. slowly.

Segment lifetime issues

The maximum time a segment can be delayed in the network is referred to as the maximum segment lifetime. This is essentially the path of longest delay and is a conservative estimate rather than a known or measurable quantity. Factors affecting the maximum segment lifetime are the presence of satellite links between packet switches and whether internetting may be encountered between two communicating TCP's.

The segment lifetime affects the TCP header size through its relationship to the sequence number space. The sequence number space must be chosen so that no two distinct octets can be assigned the same sequence number. (It is possible for identical octets with the same sequence number to be present as in the case of a retransmitted segment.) To achieve this, the sequence number space must be large enough to assure that an old octet cannot possible be in the network if its sequence number corresponds to a newly numbered octet.

A table is presented below which indicates the relationship between the TCP header size and the segment lifetime.

Resynchronization considerations

When selecting a sequence number space size, the resynchronization requirements should be considered. As shown in 9.1.2.4.12 of the body of the specification, the minimum time between resynchronizations for a connection of low activity is dependent on the sequence number space, the slope of the ISN lines, and the maximum segment lifetime. A small sequence space may require resynchronization more often than desired and therefore introduce its own overhead. If the slope of the ISN lines is decreased below the actual sequence number consumption rate, in hopes of increasing the horizontal distance between the ISN lines, waiting will frequently occur due to the vertical collision with the ISN lines during bursty transmission.

Internetting considerations

Addressing

TCP-to-TCP communication between networks requires additional addressing fields in the TCP header. The addressing fields include both source and destination addresses for the networks and the TCP's. If the internetting capability is removed, all of these fields can be deleted, since the S-segment header, the Binary Segment Leader, holds the TCP address in the Autodin II network.

Fragmentation

Since internet gateways can cause fragmentation of T-segments (into T-segments of smaller sizes), the TCP header must include bits that flag T-segment boundaries. The BOS and EOS control bits are required for the reassembly of fragmented segments at the destination. In the absence of internetting, conventions can be established for all TCP's that would eliminate fragmentation. Essentially this requires sending T-segments which are equal to or smaller than the maximum subnet packet so that fragmentation by the packet switches is not experienced.

Variable-sized headers

There is a potential savings in omitting some header fields when they are not required. This could give substantial savings in the header size, as will be shown, but the savings is not constant and at this time rather unpredictable. Also, the mechanism for omitting unnecessary fields will introduce extra header bits in the worst case and will also require extra processing to parse the header.

In one case, with port addresses, an abbreviation for the port can be substituted for the full port address once the connection has been established. The abbreviation, or index, is a replacement for both source and destination ports. It is different for each TCP and thus does not have to be chosen from a global name space. The full port addresses must be used when closing the connection to assure reliable management of ports through crashes.

The strategy for implementing variable-sized headers is to provide a bit map of the excludable fields to help in the parsing of the fields. The fields would be grouped at the end of the header to simplify parsing of the non-excludable fields. The following fields are considered to be optional in a TCP header:

Field	Bits	Frequency of use
Acknowledgement	32	usually present
Window size	16	usually present
Control-data Extension	8	rarely present
Internet fields	48	not present if local network segment
Port address	24	present for establishment and closing only

Header size vs. Byte size

The fields that could be affected by byte size are the sequence number (SN bits), acknowledgement (ACK bits), window size (WS bits), and text length (TL bits). As mentioned earlier, the text length field should not be made to refer to large byte boundaries, rather to some small unit such as the octet. Therefore, only the SN, ACK, and WS bits are of interest.

The SN and ACK sizes should be equal and related to the byte size and S, the sequence number space in numbering individual bits. The window size field reflects the largest flow control unit that can be advertised by the receiver. While it has some constraints on its maximum size as a result of the sequence number space, the size should represent the largest user receive buffer allocation for each connection. We have chosen 2**19 bits as a reasonable maximum. The window size field will be related to the byte size, so that for octets it is 2**16. The affected bits (AB) are

AB = SN + ACK + WS = (2 * SN) + (19 - Log2 (byte-size))

The sequence number field is

SN = Log2 (S / byte-size)

A table of affected bits as a function of byte size and sequence number space follows:

	Byte Size(bits)					
S	8	16	32	64	256	1024
2**35	80	77	74	71	65	59
2**27	64	61	58	55	49	43

Header size vs. Segment lifetime

The estimated maximum segment lifetime may affect the choice of sequence number space as stated in 9.1.2.4. The sequence number space size directly affects the size of the sequence number (SN) and acknowledgement fields. For convention,

S = Sequence number space = 2 ** SN (bytes)

B = Bandwidth (bits/sec) = 56,000 bits/sec

T = Segment lifetime (secs)

The sequence number space is constrained by

S > 2 * B * T / byte-size

So, for octets

S > 14,000 * T

The affected bits (AB) in the TCP header are

AB = 2 * SN = 2 * Log2 (S)

The minimum sequence number space and header size for various segment lifetimes follows:

Segment Lifetime (secs)	Min S (octets)	Min AB (bits)
2**4 (16 sec)	2**18	36
2**10 (17 min)	2**24	48
2**14 (4.5 hr)	2**28	56
2**18 (3 days)	2**32	64

APPENDIX E. TRANSMISSION CONTROL BLOCK (TCB)

The TCB contains the following information (field sizes are approximate only and may vary from one implementation to another):

16 bits: Local connection name

16 bits: Process ID

16 bits: Local port

32 bits: Foreign socket

16 bits: Receive window size in octets

32 bits: Receive left window edge (next sequence number expected)

32 bits: Receive unacknowledged left window edge (copied to user buffer but not yet ack'd)

16 bits: Receive filter (max right window edge advertised)

16 bits: Send window size in octets

32 bits: Send left window edge (earliest unacknowledged octet)

32 bits: Next send sequence number

8 bits: Connection state

8 bits: Segment retransmission timeout
(in quarters of a second)
(max retransmission timeout = 2^8*.25 = 64sec.)

16 bits: Head of from-user buffer queue

16 bits: Tail of from-user buffer queue

16 bits: Pointer to last octet segmentized in from-user buffer queue (refers to the buffer at the head of the queue)

16 bits: Head of to-net buffer queue

16 bits: Tail of to-net buffer queue

16 bits: Head of to-net control queue

16 bits: Tail of to-net control queue

16 bits: Head of Segmentized buffer queue

16 bits: Tail of Segmentized buffer queue

16 bits: Head of Retransmit queue

16 bits: Tail of Retransmit queue

16 bits: Head of to-user buffer queue

16 bits: Tail of to-user buffer queue

16 bits: Total unfilled to-user buffer space

16 bits: Head of Reassembly queue

16 bits: Tail of Reassembly queue

16 bits: Pointer to last octet reassembled
(refers to buffer at top of reassembly queue)

16 bits: Pointer to last octet filled in partly filled buffer
(refers to buffer at top of to user buffer queue)

16 bits: Pointer to next "hole" in the reassembly queue.

16 bits: Forward TCB pointer

1 bit: Connection activity flag

32 bits: WOPEN sequence number (zero if none exists)

8 bits: WOPEN timeout

4 bits: Security level of the connection

1 bit: Security level type (0 = absolute; 1 = ceiling)

4 bits: Precedence of the connection

8 bits: TCC of the connection

APPENDIX F. TCP STATES AND TRANSITIONS

A. Connection states (see Figure 7 for state diagram)

CLOSED (no TCB)

OPEN

SYNsent

SYNsent-rcvd

ESTD

CLOSErcvd

FINsent

FINsent-ackd

FINrcvd

FINsent-rcvd

DSNsent

DSNrcvd

SIMULDSN

DSNsent-FINrcvd

DSNsent-FLFINrcvd

FINsent-DSNrcvd

FINackd-DSNrcvd

B. State-Changing Controls

1. In the following sections, the state transitions associated wih the receipt of various controls are listed. The controls are listed first, with the transition from each TCP state below the control. Controls in this context include SIP-to-TCP controls, user-to-TCP controls, TCP-TCP controls and errors, and internal events in the TCP that may result in a state change.

2. SIP-to-TCP Controls

a. For the SIP-to-TCP controls below, the connection should be closed immediately by deleting the TCB, returning all buffers to the user, and notifying the user with the appropriate message.

Reject by PS

- security violation (at src)
- security violation (at dst)
- illegal precedence (at src)
- illegal precedence (at dst)
- illegal user group
- illegal address

Undelivered

- host dead
- wrong precedence
- stop transmission at destination

3. Remote-to-local TCP Controls

a. State changing controls at the TCP-TCP level consist of both normal controls and error event codes. Below we show actions and transitions for receipt of each control.

b. ACK received

OPEN
Return "connection does not exist" to remote TCP

SYNsent
Check if this acks the SYN that we sent and that a SYN was also received. If everything OK, then ack the SYN, send a message to the user, and move to the ESTD state. If the ACK does not ack our SYN, then discard.

SYNsent-rcvd
Check if this acks the SYN we sent. If it does then send a message to the user and move to the ESTD state. If it doesn't, then discard.

ESTD
Call the ACK'er to remove ack'd segments from the retransmit queue.

CLOSErcvd
  If the ACK refers to sequence numbers lower than the last sequence number assigned to data when the CLOSE was received, then treat as ESTD state.

FINsent
  Check that ACK is for the FIN we sent and move to the FINsent-ackd state.

FINsent-ackd
  Discard.

FINrcvd
  Handle as ESTD.

FINsent-rcvd
  Check that ACK is for the FIN we sent, delete the TCB, and move to the CLOSED state.

DSNsent
  If ACK is for sequence numbers lower than the DSN we sent, then handle as ESTD state. Otherwise, check that ACK is for the DSN we sent and that it uses old sequence numbers; establish new sequence numbers and send a SYN with new sequence number; and move to ESTD state.

DSNrcvd
  Discard.

SIMULDSN
  If ACK is for sequence numbers lower than the DSN we sent, then handle as ESTD state. Otherwise, check that ACK is for the DSN we sent and that it uses old sequence numbers; establish new sequence numbers and send a SYN with new sequence number; and move to DSNrcvd state.

DSNsent-FINrcvd
  Check that ACK is for the DSN we sent and that it uses old sequence numbers; establish new sequence numbers and send a SYN with new sequence number; move to FINrcvd state.

DSNsent-FLFINrcvd
  Check that ACK is for the DSN we sent and send SYN, FIN, FL, ACK(for the FIN); move to FINsent-rcvd state.

FINsent-DSNrcvd
Check that ACK is for the FIN we sent. If so, then move to the FINackd-DSNrcvd state.

FINackd-DSNrcvd
Discard.

c. SYN received

OPEN
If other control bits like FIN, ACK, INT, or DSN are on then send an EFP 6, else send a SYN / ACK(for the SYN received) and move to the SYNrcvd state.

SYNsent
If other control bits like FIN, INT, or DSN are on then send an EFP 6. If accompanied by an ACK for our SYN, then send a message to the user and move to the ESTD state, else send an ACK for the SYN and move to the SYNsent-rcvd state.

SYNsent-rcvd
If this SYN is not a duplicate of the SYN that put us in this state, then send an EFP 6 and ALSO go back to the OPEN state and try again.

ESTD
If this SYN is not a duplicate of the SYN that put us in this state, then send an EFP 6.

CLOSErcvd
Discard.

FINsent
Discard.

FINsent-ackd
Discard.

FINrcvd
Discard.

FINsent-rcvd
Discard.

DSNsent
Discard.

DSNrcvd
ACK the SYN received, set new receive sequence numbers, and move to the ESTD state.

SIMULDSN
ACK the SYN received, set new receive sequence numbers, and move to the DSNsent state.

DSNsent-FINrcvd
Discard.

DSNsent-FLFINrcvd
Discard.

FINsent-DSNrcvd
ACK the SYN received, set new receive sequence numbers, and move to the FINsent state.

FINackd-DSNrcvd
ACK the SYN received, set new receive sequence numbers, and move to the FINsent-ackd state.

d. FIN-FL (Immediate) received

OPEN
Return EFP 7 to remote TCP.

SYNsent
Return EFP 7 to remote TCP. Return to OPEN state.

SYNsent-rcvd
Return EFP 7 to remote TCP. Return to OPEN state.

ESTD
Flush user-to-net data; flush net-to-user data; send a FIN; ack the FIN received; send a message to the user; move to FINsent-rcvd state.

CLOSErcvd
Send a FIN; ACK the FIN received; flush user-to-net data; move to FINsent-rcvd state.

FINsent
Ack the FIN and move to FINsent-rcvd state.

FINsent-ackd
Ack the FIN; flush user-to-net; signal user; move to CLOSED state.

FINrcvd
Discard.

FINsent-rcvd
Discard.

DSNsent
Flush net-to-user; flush user-to-net; signal user; move to DSNsent-FLFINrcvd state.

DSNrcvd
Discard. Handle it when in an acceptable state, if it's retransmitted.

SIMULDSN
Discard. Assume SYN and ACK will put us in a state to handle this if it is retransmitted.

DSNsent-FINrcvd
Discard.

DSNsent-FLFINrcvd
Discard.

FINsent-DSNrcvd
Discard. Not believable.

FINackd-DSNrcvd
Discard. Not believable.

e. FIN (Deferred) received

OPEN
Return EFP 7 to remote TCP

SYNsent
Return EFP 7 to remote TCP. Return to OPEN state.

SYNsent-rcvd
Return EFP 7 to remote TCP. Return to OPEN state.

ESTD
  Flush user-to-net data and move to FINrcvd state.

CLOSErcvd
  Send a FIN; ACK the FIN received; flush user-to-net data; send message to user; move to FINsent-rcvd state.

FINsent
  Ack the FIN; send message to user; move to FINsent-rcvd state.

FINsent-ackd
  Ack the FIN; flush user-to-net data; send a message to the user; delete TCB; move to CLOSED state.

FINrcvd
  Discard.

FINsent-rcvd
  Discard.

DSNsent
  Flush user-to-net data and move to DSNsent-FINrcvd state.

DSNrcvd
  Discard. Handle it when in the ESTD state, if it's retransmitted.

SIMULDSN
  Discard. Assume SYN and ACK will put us in a state to handle this if it is retransmitted.

DSNsent-FINrcvd
  Discard.

DSNsent-FLFINrcvd
  Discard.

FINsent-DSNrcvd
  Discard; not believable.

FINackd-DSNrcvd
  Discard; not believable.

f. DSN received

OPEN
Return "connection does not exist" to remote TCP

SYNsent
Return "connection does not exist" to remote TCP

SYNsent-rcvd
Return connection does not exist" to remote TCP

ESTD
If everything up to the DSN has been acknowledged, send an ACK for the DSN, and move to the DSNrcvd state. Otherwise discard the DSN (it will be retransmitted).

CLOSErcvd
Discard. Connection closing.

FINsent
ACK everything up to DSN, send an ACK for the DSN, and move to the FINsent-DSNrcvd state.

FINsent-ackd
Discard. Connection closing.

FINrcvd
Discard. Connection closing.

FINsent-rcvd
Discard. Connection closing.

DSNsent
ACK everything up to DSN, send an ACK for the DSN, and move to the SIMULDSN state.

DSNrcvd
Discard.

SIMULDSN
Discard.

DSNsent-FINrcvd
Discard; already in a closing sequence, only need an ACK.

DSNsent-FLFINrcvd
Discard. Connection will close.

FINsent-DSNrcvd
  Discard.

FINackd-DSNrcvd

  Discard.

g. RESET received

OPEN
  Return "connection does not exist" to remote TCP

SYNsent
  If RESET is believable, then move to OPEN state.

SYNsent-rcvd
  If RESET is believable, then move to OPEN state.

ESTD
  If RESET is believable, then move to OPEN state; return buffers to the user and flush user-to-net.

CLOSErcvd
  If RESET is believable, then move to CLOSED state; inform user of a flushing close.

FINsent
  If RESET is believable, then move to CLOSED state; inform user of a flushing close.

FINsent-ackd
  If RESET is believable, then move to CLOSED state; inform user of a flushing close.

FINrcvd
  If RESET is believable, then move to CLOSED state; inform user of a flushing close.

FINsent-rcvd
  If RESET is believable, then move to CLOSED state; inform user of a flushing close.

DSNsent
  If RESET is believable, then move to OPEN state; return buffers to the user and flush user-to-net.

DSNrcvd
If RESET is believable, then move to OPEN state; return buffers to the user and flush user-to-net.

SIMULDSN
RESET couldn't be believable since we couldn't have generated an EFP 6 in the previous state.

DSNsent-FINrcvd
RESET couldn't be believable since we couldn't have generated an EFP 6 in the previous state.

DSNsent-FLFINrcvd
RESET couldn't be believable since we couldn't have generated an EFP 6 in the previous state.

FINsent-DSNrcvd
RESET couldn't be believable since we couldn't have generated an EFP 6 in the previous state.

FINackd-DSNrcvd
RESET couldn't be believable since we couldn't have generated an EFP 6 in the previous state.

h. Unacceptable SYN (or SYN/ACK) arrived at foreign TCP

OPEN
Discard; not believable.

SYNsent
Send a RESET and move to OPEN state.

SYNsent-rcvd
Move to OPEN state.

ESTD
Could have been generated when moving from DSNsent to ESTD, so check if it is for the SYN that we sent, if so, then move to the CLOSED state.

CLOSErcvd
Discard; not believable.

FINsent
Discard; not believable.

FINsent-ackd
Discard; not believable.

FINrcvd
May have come here from DSNsent-FINrcvd; our SYN was unacceptable, which implies that resynchronization was unsuccessful. Delete the TCB and move to CLOSED, leaving a half-open connection.

FINsent-rcvd
May have come here from DSNsent-FLFINrcvd; our SYN was unacceptable, which implies that resynchronization was unsuccessful. Delete the TCB and move to CLOSED, leaving a half-open connection.

DSNsent
Discard; not believable.

DSNrcvd
SYN could have been generated moving from SIMULDSN to here; our SYN was unacceptable, which implies that resynchronization was unsuccessful. Delete the TCB and move to CLOSED, leaving a half-open connection.

SIMULDSN
Discard; not believable.

DSNsent-FINrcvd
Discard; not believable.

DSNsent-FLFINrcvd
Discard; not believable.

FINsent-DSNrcvd
Discard; not believable.

FINackd-DSNrcvd
Discard; not believable.

i. Connection does not exist at foreign TCP

OPEN
Discard.

In all other states, send a message to the user, and simulate a FL-FIN closing to the user.

j. Foreign TCP inaccessible

OPEN
  Discard.

SYNsent
  Move to OPEN and pass message to user.

SYNsent-rcvd
  Move to OPEN and try synchronization again.

ESTD
  Move to OPEN and pass message to user.

CLOSErcvd
  Simulate a flushing close for the user.

FINsent
  Simulate a flushing close for the user.

FINsent-ackd
  Simulate a flushing close for the user.

FINrcvd
  Simulate a flushing close for the user.

FINsent-rcvd
  Simulate a flushing close for the user.

DSNsent
  Move to OPEN and pass message to user.

DSNrcvd
  Move to OPEN and pass message to user.

SIMULDSN
  Move to OPEN and pass message to user.

DSNsent-FINrcvd
  Simulate a flushing close for the user.

DSNsent-FLFINrcvd
  Simulate a flushing close for the user.

FINsent-DSNrcvd
  Simulate a flushing close for the user.

FINackd-DSNrcvd
  Simulate a flushing close for the user.

k. Security violation at destination TCP

OPEN
  Discard.

In all other states, pass message to user and simulate a FL-FIN closing for the user.

l. Precedence or TCC violation at destination TCP

OPEN
  Discard.

SYNsent
  Pass message to user and move to OPEN state.

SYNsent-rcvd
  Pass message to user and move to OPEN state.

ESTD
  Pass message to user and move to OPEN state.

CLOSErcvd
  Pass message to user and simulate a flushing close for the user.

FINsent
  Pass message to user and simulate a flushing close for the user.

FINsent-ackd
  Pass message to user and simulate a flushing close for the user.

FINrcvd
  Pass message to user and simulate a flushing close for the user.

FINsent-rcvd
  Pass message to user and simulate a flushing close for the user.

DSNsent
  Shouldn't happen; pass message to user and simulate a flushing close.

DSNrcvd
  Pass message to user and simulate a flushing close for the user.

SIMULDSN
  Shouldn't happen; pass message to user and simulate a flushing close.

DSNsent-FINrcvd
  Pass message to user and simulate a flushing close for the user.

DSNsent-FLFINrcvd
  Pass message to user and simulate a flushing close for the user.

FINsent-DSNrcvd
  Pass message to user and simulate a flushing close for the user.

FINackd-DSNrcvd
  Pass message to user and simulate a flushing close for the user.

4. User-to-TCP

a. OPEN

CLOSED (no TCB)
  Move to the OPEN state.

ALL OTHER STATES
  Return "connection already open" to the user.

b. CLOSE (Deferred)

CLOSED (no TCB)
  Reject with "connection not open" message.

OPEN
Delete TCB and associated data structures; return all buffers to the user; send "connection closing" message to the user.

SYNsent
Delete TCB and associated data structures; return all buffers to the user; send "connection closing" message to the user; send "connection closing" message to the remote TCP.

SYNsent-rcvd
Delete TCB and associated data structures; return all buffers to the user; send "connection closing" message to the user; send "connection closing" message to the remote TCP.

ESTD
Flush the net-to-user data; return the user's receive buffers; move to the CLOSErcvd state.

CLOSErcvd
Reject with "connection not open."

FINsent
Reject with "connection not open."

FINsent-ackd
Reject with "connection not open."

FINrcvd
Send a FIN and an ACK for the FIN received; move to the FINsent-rcvd state.

FINsent-rcvd
Reject with "connection not open."

DSNsent
Defer action until in the ESTD state.

DSNrcvd
Defer action until in the ESTD state.

SIMULDSN
Defer action until in the ESTD state.

DSNsent-FINrcvd
Defer action until in the FINrcvd state.

DSNsent-FLFINrcvd
Discard; connection is closing.

FINsent-DSNrcvd
Reject with "connection not open."

FINackd-DSNrcvd
Reject with "connection not open."

c. *CLOSE (Immediate)*

CLOSED (no TCB)
Reject with "connection not open" message.

OPEN
Delete TCB and associated data structures; return all buffers to the user; send "connection closing" message to the user.

SYNsent
Delete TCB and associated data structures; return all buffers to the user; send "connection closing" message to the user; send "connection closing" message to the remote TCP.

SYNsent-rcvd
Delete TCB and associated data structures; return all buffers to the user; send "connection closing" message to the user; send "connection closing" message to the remote TCP.

ESTD
Flush the net-to-user data; flush the user-to-net data; return the user's receive and send buffers; send FIN and FL; move to the FINsent state.

CLOSErcvd
Reject with "connection not open."

FINsent
Reject with "connection not open."

FINsent-ackd
Reject with "connection not open."

FINrcvd
Send a FIN and an ACK for the FIN received; flush net-to-user data; move to the FINsent-rcvd state. (message to user ???)

FINsent-rcvd
Reject with "connection not open."

DSNsent
Defer action until in the ESTD state.

DSNrcvd
Defer action until in the ESTD state.

SIMULDSN
Defer action until in the ESTD state.

DSNsent-FINrcvd
Defer action until in the FINrcvd state.

DSNsent-FLFINrcvd
Discard; connection is closing.

FINsent-DSNrcvd
Reject with "connection not open."

FINackd-DSNrcvd
Reject with "connection not open."

5. Internally Generated

a. Synchronization timeout (timer set when moving from OPEN state to a SYNxxx)

CLOSED (no TCB)
Not believable.

OPEN
Not believable.

SYNsent
Reset connection by resetting the TCB parameters; send timeout message to the user; return send and receive buffers to user; move to the OPEN state.

SYNsent-rcvd
Reset connection by resetting the TCB parameters; send

timeout message to the user; return send and receive buffers to user; move to the OPEN state.

ESTD
Not believable.

CLOSErcvd
Not believable.

FINsent
Not believable.

FINsent-ackd
Not believable.

FINrcvd
Not believable.

FINsent-rcvd
Not believable.

DSNsent
Not believable.

DSNrcvd
Not believable.

SIMULDSN
Not believable.

DSNsent-FINrcvd
Not believable.

DSNsent-FLFINrcvd
Not believable.

FINsent-DSNrcvd
Not believable.

FINackd-DSNrcvd
Not believable.

b. Segment retransmit count exceeded

CLOSED (no TCB)
Not believable. Ignore.

OPEN
Not believable. Ignore.

SYNsent
Not believable. Ignore.

SYNsent-rcvd
Not believable. Ignore.

ESTD
Check the send window size. If it is zero, then ignore. If it is nonzero, then send a FIN and "connection closing" to the remote TCP and send "foreign TCP inaccessible" to the local user.

CLOSErcvd
Check the send window size. If it is zero, then ignore. If it is nonzero, then send a FIN and "connection closing" to the remote TCP and send "foreign TCP inaccessible" to the local user.

FINsent
Check the send window size. If it is zero, then ignore. If it is nonzero, then send a FIN and "connection closing" to the remote TCP and send "foreign TCP inaccessible" to the local user.

FINsent-ackd
Check the send window size. If it is zero, then ignore. If it is nonzero, then send a FIN and "connection closing" to the remote TCP and send "foreign TCP inaccessible" to the local user.

FINrcvd
Check the send window size. If it is zero, then ignore. If it is nonzero, then send a FIN and "connection closing" to the remote TCP and send "foreign TCP inaccessible" to the local user.

FINsent-rcvd
Check the send window size. If it is zero, then ignore. If it is nonzero, then send a FIN and "connection closing" to the remote TCP and send "foreign TCP inaccessible" to the local user.

DSNsent
Check the send window size. If it is zero, then ignore.

If it is nonzero, then send a FIN and "connection closing" to the remote TCP and send "foreign TCP inaccessible" to the local user.

DSNrcvd
Check the send window size. If it is zero, then ignore. If it is nonzero, then send a FIN and "connection closing" to the remote TCP and send "foreign TCP inaccessible" to the local user.

SIMULDSN
Check the send window size. If it is zero, then ignore. If it is nonzero, then send a FIN and "connection closing" to the remote TCP and send "foreign TCP inaccessible" to the local user.

DSNsent-FINrcvd
Check the send window size. If it is zero, then ignore. If it is nonzero, then send a FIN and "connection closing" to the remote TCP and send "foreign TCP inaccessible" to the local user.

DSNsent-FLFINrcvd
Check the send window size. If it is zero, then ignore. If it is nonzero, then send a FIN and "connection closing" to the remote TCP and send "foreign TCP inaccessible" to the local user.

FINsent-DSNrcvd
Check the send window size. If it is zero, then ignore. If it is nonzero, then send a FIN and "connection closing" to the remote TCP and send "foreign TCP inaccessible" to the local user.

FINackd-DSNrcvd
Check the send window size. If it is zero, then ignore. If it is nonzero, then send a FIN and "connection closing" to the remote TCP and send "foreign TCP inaccessible" to the local user.

c. Connection preemption

Handle as an Immediate CLOSE from the user.

d. Resynchronize sequence numbers

CLOSED (no TCB)
Not believable. Ignore.

OPEN
Not believable. Ignore.

SYNsent
Not believable. Ignore.

SYNsent-rcvd
Not believable. Ignore.

ESTD
Send a DSN with old sequence numbers and move to the DSNsent state.

CLOSErcvd
Must have sequence space enough for one FIN, so ignore the need to resynchronize.

FINsent
Connection can close without resynchronization, so ignore it.

FINsent-ackd
Connection can close without resynchronization, so ignore it.

FINrcvd
Must have sequence space enough for one FIN, so ignore the need to resynchronize.

FINsent-rcvd
Connection can close without resynchronization, so ignore it.

DSNsent
Not believable. Ignore.

DSNrcvd
Send a DSN using old sequence numbers and move to the SIMULDSN state.

SIMULDSN
Not believable. Ignore.

DSNsent-FINrcvd
  Not believable. Ignore.

DSNsent-FLFINrcvd
  Not believable. Ignore.

FINsent-DSNrcvd
  Connection can close without resynchronization, so ignore it.

FINackd-DSNrcvd
  Connection can close without resynchronization, so ignore it.

e. User process death

CLOSED (no TCB)
  No action.

OPEN
  Delete TCB; remove any user buffers from TCP's queues.

SYNsent
  Delete TCB; send FIN and "connection closing" to remote TCP; remove any user buffers from TCP's queues.

SYNsent-rcvd
  Delete TCB; send FIN and "connection closing" to remote TCP; remove any user buffers from TCP's queues.

ESTD
  Delete TCB; send FIN and "connection closing" to remote TCP; remove any user buffers from TCP's queues.

CLOSErcvd
  Delete TCB; send FIN and "connection closing" to remote TCP; remove any user buffers from TCP's queues.

FINsent
  Delete TCB; send FIN and "connection closing" to remote TCP; remove any user buffers from TCP's queues.

FINsent-ackd
  Delete TCB; send FIN and "connection closing" to remote TCP; remove any user buffers from TCP's queues.

FINrcvd
Delete TCB; send FIN and "connection closing" to remote TCP; remove any user buffers from TCP's queues.

FINsent-rcvd
Delete TCB; send FIN and "connection closing" to remote TCP; remove any user buffers from TCP s queues.

DSNsent
Delete TCB; send FIN and "connection closing" to remote TCP; remove any user buffers from TCP's queues.

DSNrcvd
Delete TCB; send FIN and "connection closing" to remote TCP; remove any user buffers from TCP's queues.

SIMULDSN
Delete TCB; send FIN and "connection closing" to remote TCP; remove any user buffers from TCP's queues.

DSNsent-FINrcvd
Delete TCB; send FIN and "connection closing" to remote TCP; remove any user buffers from TCP's queues.

DSNsent-FLFINrcvd
Delete TCB; send FIN and "connection closing" to remote TCP; remove any user buffers from TCP's queues.

FINsent-DSNrcvd
Delete TCB; send FIN and "connection closing" to remote TCP; remove any user buffers from TCP's queues.

FINackd-DSNrcvd
Delete TCB; send FIN and "connection closing" to remote TCP; remove any user buffers from TCP's queues.

APPENDIX G. FLOW CONTROL WINDOW SIZE SETTING

A. Let B denote the total bandwidth of the TCP, i.e. the rate at which a TCP can consume data from the network provided enough user resources are available. From this bandwidth we assign a bandwidth of BI for each active Category I connection, leaving whatever remaining bandwidth to be used by lower precedence connections. In addition we define the following:

WLTI - The long term window for Category I connections.

WLT - The long term window for non-Category I connections.

NI - The number of established Category I connections in the TCP

N - the number of non-Category I connection in the TCP

WI - The current window for a Category I connection

W - The current window for a non-Category I connection

B. Using these definitions we assign:

```
WLTI = BI

WLT = MAX( (B-BI*NI)/N , 0 )
```

C. The algorithm used to update the window is the following: Upon the processing of a user's RECEIVE the window is set to

```
W = MIN( WLT , Total available user space )

or

WI = MIN( WLTI , Total available user space )
```

D. And upon every segment sent on the network from this connection the to-net segment handler sets the window to

```
W = 0.5*( WLT*Available/Total + W )

or

WI = 0.5*( WLTI*Available/Total + WI )
```

E. This model meets the criteria set in the main document (see Flow Control) as follows:

1. WLT is dependent upon the number of the connections, thereby administering fairness among connections.

2. The window size will never exceed the bandwidth allocated to this connection. The algorithm may sometimes give credit to a "well behaving" user by setting his window greater than the actual buffer available. This window will be reduced if the user does not issue new RECEIVEs promptly.

3. The current window size is dependent upon previous window sizes and upon the rate at which the user makes letter space available. If a user fails to make such space available, his window will be cut by a factor of two every time a segment is sent from this side of the connection. (The TCP may also apply a threshold mechanism by which a window is set to zero when it is reduced below the threshold.)

4. The algorithm supports high throughput for high precedence connections. Note that WLT (but never WLTI) may be reduced to zero, forcing a zero window for all low precedence connections, if several high precedence connections are active. (The number of high precedence connections that will force zero window on the rest is determined by the B/BI ratio.)

APPENDIX H. ZERO FLOW CONTROL WINDOWS

A. Setting the window to zero

1. Setting a window to zero to stop data flow can be done by sending a segment with zero in the window size field. This informs the foreign TCP that the local TCP's receive window is closed.

2. The from-net segment handler algorithm for interpreting segments must allow window size changes on segments that acknowledge old duplicates. The window size field of a received segment with an ACK field equal to the receive left window edge is always used to update the current send window size. This is essential for setting the local receive window to zero in the face of new data arrivals when there is no data to carry the new window size. A receiving TCP does not want to generate an ACK that acknowledges any of the new data so it sends an ACK with its send left window edge in the ACK field.

B. Opening a zero window

1. Opening a window of size zero presents some special problems. Since a window size can accompany each segment, it seems that the normal data segment and acknowledgement transmissions should be sufficient to vary the size of the windows. However, when the remote TCP is showing a zero receive window it is difficult to send a window change reliably. If ACKs are used and they arrive out-of-order, it may be impossible to tell if the window is opening or closing.

2. The problem of opening a window of size zero is solved by using a pair of control segments, one sent by the local TCP that is making its window size nonzero (WOPEN) and one that is sent by the foreign TCP to acknowledge the opening (WACK).

3. The WOPEN is sent with the next available sequence number, i.e. one that has never been used for sending. It does not consume the sequence number; the sequence number is used merely as a name for this WOPEN. The ACK bit on the WOPEN segment is set to zero and the window size represents the new nonzero receive window for the local TCP. No data or other control can accompany this segment since it makes unique use of the sequence number. The sequence number of the WOPEN is stored in the local TCB, along with some timeout data for retransmitting the WOPEN. Only one WOPEN can be outstanding at one time, so further

attempts to internally change from a zero to nonzero window are ignored.

4. At the foreign TCP, the receipt of a WOPEN is guaranteed in much the same way as the receipt of a FIN. Once received, no ACK is generated for the WOPEN. The foreign send window is udpated in the foreign TCP's TCB. The WOPEN is acknowledged with a WACK, which uses the foreign TCP's next available sequence number, but does not consume it. The WACK has the ACK bit set to zero and the sequence number it received in the WOPEN in the acknowledgement field. The window field holds the current foreign receive window. A WOPEN received when the foreign send window is nonzero, causes the same action as a WOPEN for a zero window; however, a WOPEN can never have a zero window.

5. Whether or not a segment contains a WOPEN control, the window size of the highest numbered segment should be used for flow control information.

6. Receipt of the WACK at the local TCP causes the sequence number and timeout data in the TCB to be deleted. However, the local TCP can accept data prior to the WACK. It is acceptable to allow receipt of data as soon as the TCB field for the WOPEN is set. In this case the record of the WOPEN in the TCB can be cleaned up immediately and subsequent WACK's discarded. Any WACK received that does not correspond exactly to the WOPEN sequence number in the TCB will be discarded.

C. Sending to a zero window

1. A zero receive window indicates an unwillingless to receive data, most likely because there are no user receive buffers into which data can be placed. TCP must perform special functions with regard to sending segments in this case. If no data is being sent on the connection, a zero window is of no concern. However, if data is queued for sending and the window remains zero for longer than some timeout period, the connection will be closed. The timeout period should be set for 60 seconds initially. In addition to this watchdog function, the TCP should notify the user (at 15 second intervals) with a message indicating the unwillingness of the remote user to accept data.

2. The retransmission algorithm should suspend retransmission when a receive window of size zero is found in the remote TCP, except for the following controls: INT-FL, FL, FIN-FL, SYN, WOPEN.

APPENDIX I. SCCU

A. Introduction

1. This appendix describes modifications and changes to be made in the TCP implementation for an SCCU. The appendix essentially traverses the main document and points out the places where changes should be made. The reader of the appendix is assumed to be acquainted with the functions and implementation model of the TCP that is presented in the main document. Much of what follows are recommendations for SCCU TCP implementation. Individual implementations may include more of what is suggested here up to a full scale TCP.

2. The underlying assumptions for a SCCU TCP implementation are that only one connection may exist at any one time, and that the TCP has to look, from the network standpoint, as any other TCP.

3. Much of the savings in SCCU TCP implementation is a result of these assumption. Introducing a second connection (e.g., for preemption purposes at higher level protocol such as THP) requires the inclusion of the connection multiplexing mechanism, multiple connection bookkeeping etc. This will make the SCCU less unique in its structure, and in fact more similar to a TAC. SCCU's are envisioned to support mainly process-to-process communication in lieu of a private line, and not as a general interface to the network. If more than one connection, or a more general interface are needed, it is recommended that the interface be via a TAC so the number of connections is limited only by the availability of resources.

B. Background

1. Configuration Overview

a. The Single Channel Control Unit (SCCU), provides a single-connection interface to the network. The principal intended use of the SCCU is to provide to a pair of hosts currently connected via a private line the less expensive alternative of using the network. In this situation each host would have an SCCU interface to the network. However, provisions are made for non-SCCU hosts to communicate with SCCU hosts and vice versa.

b. Two types of SCCU hosts are envisioned: the first interfaces the user's process to the TCP via a THP module, the other via a Host Specific Interface (HSI). Although the capabilities of both configurations are the same, the first is intended to enable terminal oriented users of one host to appear as local (terminal) users to the other host. The second configuration is used when the raw connection is employed by a process for a more general activity.

2. Protocol Overview

a. As the SCCU is an integral part of the AUTODIN II network the need for the various protocol levels and functions still exist. The main task of the TCP implementer on a SCCU is to design a module that looks like any other TCP interface on the network side and has a subset of full-scale TCP from the user's side. The user behind a SCCU still expects the TCP to provide her with reliable transmission, which involves flow control, ordering and segmentizing, and notification of any relevant information about her activity

b. The important points to remember when implementing a SCCU TCP are: (1) Only one connection may exist at any one time, (2) The TCP has to look, from the network standpoint, as any other TCP and, (3) Preemption may not occur.

C. TCP Function

1. As mentioned above the function of the TCP does not change for the SCCU. The TCP has to appear like any other TCP to the network, and compromises that are made, in design and implementation, may affect only the way the TCP looks to its own user.

2. Connection Management

a. The connection is the basic tool with which processes communicate over the network; a connection in the SCCU, in this respect, is no exception. It is important that the user has a consistent conceptual view of connections, whether on a SCCU or not. The internal representation of a connection may however, be different for the SCCU case.

b. To provide for uniform addressing across the network the concept of socket was introduced. A socket is a concatenation of a TCP identifier and a PORT name, both having a prescribed syntax which the SCCU has to follow. However, since only one connection may exist in a SCCU at any one time, the port name may be selected independently by the process using the connection, or alternately provided by the SCCU without the process even knowing what it is. The LCN which is provided as a shorthand for process-TCP communication is superfluous as the connection in question is always known to the TCP.

3. Security, Precedence, and closed user groups

a. Security, precedence, and closed user groups rules are network-wide and thus have to be obeyed by the SCCU. In some SCCU implementation there will be only one process, and the same process always, that will be using the network, probably for some routine activity. If this is the case, then most of the S/P/T (especially the TCC) information may be canned into the TCP without the user having to supply them every time. Specifically, the use and complexity of the authorization table is reduced considerably.

D. TCP Environment

1. The TCP has to communicate with the process it serves and with the operating system under which it runs. Part of the TCP's environment is its interface with its neighbors, the SIP on the network side, and the user process (THP or other) on the user's side. The interaction between the TCP and the SIP should not be modified, so that the SCCU TCP appears to the network as a regular TCP. Internally however, the environment is quite different--The operating system in the SCCU is not expected to offer all the services that a regular operating system does, and the TCP-user communication should reflect the changes in the inter-process relationship.

2. Process Structure

a. Since only one process may communicate over the network at any one time, it is possible to have the TCP run as the user's process or sub-process. This configuration is advantageous as it simplifies problems of address space sharing and user-TCP communication. The implementer should be aware however, that implementing a TCP as a completely synchronous user process may create excessive idle time due to network delays. The decision on how to structure the processes is left open, and should be determined for each SCCU implementation separately.

3. Inter-process Communication

a. Although process structure in the SCCU may be different from the host or TAC cases, some asynchronous signalling mechanism must be provided. This stems from the concern of creating excessive idle time in the SCCU. If a complete asynchronous model cannot be implemented, provisions ought to be made so that the inbound part of the TCP (the from-net segment handler and the reassembler), and the entire SIP are able to run independently. This will insure that the user does not have to wait (maybe indefinitely) until all segments of incoming letters arrive, and that the TCP will be able to process incoming controls (especially those that are processed out of line) independently of the user's activity.

E. TCP Interfaces

1. The TCP interfaces with the SIP on the network side, and a user process (THP, HSI, or other) on the user's side. In most cases the behavior of the TCP on those interfaces is identical for the SCCU and non-SCCU cases. The few exceptions are outlined below.

2. TCP-USER Interface

a. Since a SCCU user and a non-SCCU user need the same kind of capabilities, essentially the same set of primitives is offered by the TCP. However, because of the special configuration, some of those primitives need fewer or different parameters, and some are used in a more restrictive manner.

b. Primitives for USER-TCP communication

i. OPEN CONNECTION

The main feature of a SCCU is that it supports only one connection at a time. As a consequence, attempts to open a second connection will be rejected with the appropriate message.

The TCP on a SCCU will support partially specified connections, i.e. a CONNECT and a LISTEN. Although never used by the user, a local port name must be associated with each connection; it may however, be chosen by the TCP rather then the process itself. For a listening connection the port should be chosen by the user so he can listen for calls on a well known socket. Local connection names are no longer necessary since the TCP and the user know exactly which connection is active.

ii. SEND LETTER

To allow for a useful and resourceful implementation, it is recommended that a SCCU TCP handle only one letter at a time. This will reduce significantly the space requirements for the TCP for both its data structures and buffer needs without considerable loss of efficiency on the user's part. The rest of this appendix assumes that this is the case. No assumption is made however, on the number of incoming letters.

An attempt to SEND a letter before the previous one is fully acknowledged will result in an appropriate error message.

iii. MOVE CONNECTION

Depending upon the process structure in which the TCP is implemented in a SCCU, this primitive may not be offered. If the TCP runs as a sub-process of the process that uses the connection, this command is meaningless. It should be noticed that this command does not create any network activity, nor does it consume any resources; it may therefore be implemented even if its use is quite rare.

3. TCP-SIP Interface

a. The TCP-SIP boundary is the network side boundary of the TCP. In order for the SCCU TCP to appear to the network as a regular TCP it is important to keep the behavior of the TCP on this boundary identically for all TCP's. This implies that the SIP on the SCCU is completely identical, in data structures and event processing, to any other SIP on the network. Consequently, the set of primitives utilized by the TCP and SIP for their mutual communication, is identical in the SCCU and non-SCCU cases.

F. TCP Implementation

1. Implementing a SCCU TCP should follow the guidelines set forth in the main document. However, the assumptions made in the previous sections should lead to a more resourceful implementation. The changes in implementation stem mainly from the fact that only one connection exists; so everything that is needed in the regular TCP to maintain multiple connections can be eliminated. Other changes reflect the additional assumptions made throughout this appendix.

2. Ordering of events should become easier as no preemption may occur and events do not have to be ordered according to precedence. Event handling will also be simpler reflecting the process structure within the SCCU.

3. TCP Data Structures

a. Inter-module Communication Support

i. Independent of the process structure that is implemented, a mechanism for inter-module communication must be supported. Events are received from the network asynchronously and must be recorded and ordered according to the internal TCP priority. Ordering events is simpler since only one connection exists; this may in fact be performed by the from-net event handler rather than the network event receiver (in which case the latter can be eliminated). The intra-TCP signal board may have fewer entries, as some of the internal calls may become synchronous.

b. Queue Descriptions

i. The main purpose of the queues is to provide the various modules of the TCP with access to letter and segment buffers. This task is not eliminated in the SCCU case. However, some queues will no longer be needed and some will reduce to one entry, eliminating the need for queue structure. The following describes the changes in the queues and queue elements.

ii. from-user buffer queue

Since a user may have only one letter outstanding, this buffer will have only one entry; thus the queue itself may be replaced by a single pointer to the user's buffer, residing in the TCB.

iii. segmentized buffer queue

In the regular TCP this queue holds completely segmentized letters. Its main purpose is to keep information about the original segments so that resegmentizing may be carried out in case of preemption. Since preemption cannot occur in the SCCU, and resegmentizing will never happen this queue is no longer needed.

iv. from-net segment queue

This is the only queue, in the regular TCP, that is per TCP and not per connection. It is used as a temporary record of buffers received from the SIP before being processed for controls and moved to the reassembly queue. Since in the SCCU case per TCP and per connection are identical properties, this buffer can also serve as the reassembly queue. This also simplifies the problem of padding the reassembly queue for segments that contain controls only.

v. reassembly queue

As indicated in the description of the from-net segment queue this queue is no longer needed.

c. TCB

i. The following entries of the TCB may be deleted:

LCN
process ID (implementation dependent)
head and tail of from-user buffer queue
head and tail of the reassembly queue
head and tail of from-net segment queue
forward TCB pointer

ii. The following entry should be added to the TCB:

pointer to user letter buffer

d. Functional Modules

i. The unique features of the SCCU lead not only to simplification and reduction in data structures, but also to easing the tasks of the various internal modules. The functions to be fulfilled by the TCP remain the same, but the fact that less data structures have to be examined and manipulated, and the fact that some situations possible in a regular TCP cannot happen in a SCCU, result in a simpler set of functional modules.

ii. Most of the functional modules described in the main document are still needed for the SCCU TCP. In the following paragraphs the changes to, and modification of, the operations of the modules are outlined.

iii. External Event Handler

Generally, the functions of this module do not change. However, since the communication mechanism with the user may change, operations on the TCP-user interface might become simpler. In addition, the external event handler does not have to sort the events according to connection and precedence.

iv. From-net Segment Handler and Reassembler

Since the from-net segment queue and the reassembly queue have been merged, it is useful to have the from-net segment handler and the reassembler merged as well. This will insure that all operations, in the incoming direction, are carried out in order. The new module will, upon the arrival of a segment, process all controls that need immediate attention, and then proceed with reassembling text if necessary.

The main problem that arises if those modules are not merged is the race condition that may occur--forcing the TCP to keep track of which controls have already been processed and what segments may be acknowledged. In general more communication between the modules will be necessary and the scheduler will have to employ a more careful algorithm in the mutual scheduling of these modules. These problems are avoided by the merge.

v. Space Manager

The operation of the space manager is much alleviated as no preemption may occur; the space manager will never have to recall resources it has previously allocated, but will always wait for these resources to be returned.

e. Connection Management

i. The SCCU TCP provides its user with the full capabilities of connection management. The user is not assumed to have any knowledge, and is unable to interfere in this task.

ii. As part of connection management the SCCU TCP supports all kinds of OPEN (CONNECT, LISTEN, and fully specified), both kinds of CLOSE (IMMEDIATE and DEFERRED), and S/P/T monitoring.

iii. Connection management in the SCCU is somewhat simpler than in the general case. It is a consequence of the fact that only one connection exists and could never be preempted. This manifests itself in simpler buffer allocation strategy since administering fairness is limited to the single existing connection. Window control should be such that the user gets the entire TCP bandwidth limited only by the resources made available to the TCP.